Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

NEH aplicada al problema de programación de la producción con dos agentes y objetivo makespan

Autor: Carmen Feliciana Fernández Márquez
Tutor: Paz Pérez González

Dpto. Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Grado Ingeniería de las Tecnologías Industriales

NEH aplicada al problema de programación de la producción con dos agentes y objetivo makespan

Autor:
Carmen Feliciana Fernández Márquez

Tutor:
Paz Pérez González

Dpto. Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Carrera: NEH aplicada al problema de programación de la producción con dos agentes y objetivo makespan

Autor: Carmen Feliciana Fernández Márquez

Tutor: Paz Pérez González

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mis padres

Agradecimientos

A José Ángel y María del Carmen, mis padres, gracias por apoyarme, no solo durante la redacción de este TFG, sino también durante toda la carrera. Gracias por confiar siempre en mí y darme ese empujón que necesitaba para creer en mi misma. Gracias a Rodrigo por enseñarme que no todo es estudiar, que hay que disfrutar la vida. Y a José Ángel por nunca mentirme sobre las dificultades que iba a entrañar la carrera y sobre todo por ser mi inspiración. Y gracias mamá por enseñarme a ser una mujer luchadora.

Gracias a mis amigos de Córdoba, por seguir ahí y por la desconexión que solo encuentro con vosotros. Gracias por enseñarme que cada uno a su manera puede lograr grandes cosas.

Gracias a mis amigos de la facultad, por nunca dejarme comer sola en la Escuela. Y también por ser mi pilar en Sevilla, habéis hecho que sienta que también tengo un hogar en esta ciudad. Gracias por todos los buenos ratos, ya fuera en Londres, en el campo, en la playa o en el escalón de la Puerta 7.

Y en general a la Escuela, por la inspiración que he encontrado entre sus paredes. Y también por enseñarme a luchar por mis sueños, por muy arduo que fuera el camino.

Carmen Feliciana Fernández Márquez

Sevilla, 2020

Resumen

Los problemas de Programación de la Producción son muy importantes en el mundo actual. Se puede decir que están presentes en todos los fundamentos de la industria moderna, de ahí la importancia de que estos sean óptimos.

El presente proyecto se centra en el modelado y resolución de un problema de programación para dos conjuntos de trabajos procesados en un sistema de *m* máquinas en serie. En el desarrollo de este proyecto se adaptarán heurísticas constructivas ya conocidas para encontrar secuencias que proporcionen un tiempo de finalización de los trabajos de un único conjunto lo más reducido posible. La optimización de este objetivo estará sujeta a una serie de restricciones. Se presentarán varias adaptaciones para poder resolver el problema y posteriormente poder analizar y comparar los resultados obtenidos.

Abstract

Scheduling problems are very important in today´s word. It can be said to be present in all the basics of modern industry, hence the importance that these are optimal.

This project focuses on modelling and solving a programming problem for two sets of jobs processed in a system of m machines in series. In the development of this project, already known constructive heuristics will be adapted to find sequences that will provide the shortest possible completion time of the work on a single set. The optimisation of this objective will be subject to a number of constraints. Several adaptations will be presented to solve the problem aas well as to analyze and compare the results obtained.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xiv**

Índice de tablas **xvi**

Índice de Figuras **xviii**

1 Introducción **1**
1.1. Contexto del Proyecto *1*
1.2. Objeto del Proyecto *2*
1.3. Estructura del documento *3*

2 Introducción a la secuenciación **5**
2.1. Notación de Graham *6*
2.2. Secuenciación de varios conjuntos de trabajos *8*

3 Instancias del problema **9**
3.1 Instancias de Taillard *9*
3.2 Generación del parámetro épsilon *10*
3.2.1. Algoritmo NEH 10
3.2.2. Adaptación algoritmo NEH para generar el valor de épsilon 14
3.2.3. Adaptación algoritmo NEHFF para generar el valor de épsilon 15
3.2.4. Valores de épsilon 20

4 Resolución del problema **23**
4.1 Adaptaciones de la heurística NEH *23*
4.1.1 NEH adaptada con mecanismo de desempate FS, NEH_{FS} 23
4.1.2 NEH adaptada con un mecanismo de desempate propuesto, $NEH_{propuesta}$ 25
4.1.3 NEH adaptada con mecanismo de desempate FF, NEH_{FF} 27
4.1.4. Función Reparación 30
4.2 Experimentación computacional *32*
4.2.1 Comparación de los algoritmos con respecto de α 33
4.2.2 Comparación de los algoritmos con respecto de m y n 36

5 Conclusiones **43**

Referencias **45**

Glosario **47**

Anexos **49**
Anexo A *49*
Código algoritmo NEH adaptado a dos grupos de trabajos para generar épsilon 49
Anexo B *58*
Código algoritmo NEHFF aplicado a dos grupos de trabajos para generar épsilon 58
Anexo C *75*

Código algoritmo NEH adaptado a dos grupos de trabajos con regla de desempate FS, NEH_{FS} 75
Anexo D *84*
Código algoritmo NEH adaptado a dos grupos de trabajos con la regla de desempate propuesta, $NEH_{PROPUESTA}$ 84
Anexo E *93*
Código algoritmo NEH adaptado a dos grupos de trabajos con la regla de desempate FF, NEH_{FF} 93
Anexo F *102*
Código Función Reparación 102

ÍNDICE DE TABLAS

Tabla 3.1: Distribución de máquinas y trabajos en las Instancias de Taillard 10

Tabla 3.2: Matriz de tiempos de proceso de los trabajos Jj en la máquina Mi 11

Tabla 3.3: Valores medios del makespan de B en la generación de épsilon 21

Tabla 4.1: Desviación porcentual relativa promedio de las diferentes heurísticas en función del valor de α y del tamaño de instancia 34

Tabla 4.2: Valores de ARPDj totales y porcentaje de factibilidad de las heurísticas en función de α 36

ÍNDICE DE FIGURAS

Figura 3.1: Diagrama de Gantt para la secuencia (J1, J4), con makespan 17 11
Figura 3.2: Diagrama de Gantt para la secuencia (J4, J1), con makespan 19 12
Figura 3.3: Diagrama de Gantt para la secuencia (J3,J1, J4), con makespan 21 12
Figura 3.4: Diagrama de Gantt para la secuencia (J1,J3, J4), con makespan 21 12
Figura 3.5: Diagrama de Gantt para la secuencia (J1,J4, J3), con makespan 20 13
Figura 3.6: Diagrama de Gantt para la secuencia (J2,J1,J4, J3), con makespan 23 13
Figura 3.7: Diagrama de Gantt para la secuencia (J1,J2, J4, J3), con makespan 24 13
Figura 3.8: Diagrama de Gantt para la secuencia (J1, J4, J2, J3), con makespan 23 14
Figura 3.9: Diagrama de Gantt para la secuencia (J1, J4, J3, J2), con makespan 23 14
Figura 3.10. Adaptación algoritmo NEH para determinar el valor de épsilon 15
Figura 3.11: Adaptación algoritmo NEHFF para generar el valor de épsilon 20
Figura 3.12: Organización de los grupos de trabajos para $NEH_{adaptada}$ y $NEHFF_{adaptada}$ 20
Figura 4.1. NEH adaptada con mecanismo de desempate FS, NEHFS 25
Figura 4.2. NEH adaptada con un mecanismo de desempate propuesto, NEHpropuesta 27
Figura 4.3. NEH adaptada con mecanismo de desempate FF, NEHFF 29
Figura 4.4. Código Función Reparación 31
Figura 4.5: Desviación porcentual relativa de NEH implementada con mecanismos de desempate para α =0 35
Figura 4.6: Variación de $ARPD_j$ con respecto al parámetro α y a los mecanimos de desempate 35
Figura 4.7: Porcentaje de factibilidad en función de α, para cada mecanismo de desempate 36
Figura 4.8: Valores ARPDj para los valores de α en función del número de máquinas, m, para NEH_{FS} 37
Figura 4.9: Valores ARPDj para los valores de α en función del número de máquinas, m, para $NEH_{PROPUESTO}$ 37
Figura 4.10: Valores ARPDj para los valores de α en función del número de máquinas, m, para NEH_{FF} 38
Figura 4.11: Valores ARPDj para los valores de α en función del número de trabajos, n, para NEH_{FS} 38
Figura 4.12: Valores ARPDj para los valores de α en función del número de trabajos, n, para $NEH_{PROPUESTA}$ 39
Figura 4.13: Valores ARPDj para los valores de α en función del número de trabajos, n, para NEH_{FF} 39
Figura 4.14: Número de instancias sin solución factible en función del número de máquinas, m 40
Figura 4.15: Número de instancias sin solución factible en función del número de trabajos, n 41

1 INTRODUCCIÓN

Para la realización de un trabajo, un industrial prevé tanto el trabajo de sus obreros como las herramientas que emplea.

- David Hume -

Para comprender el desarrollo, y sobre todo el objetivo del proyecto, en este primer capítulo se hará una contextualización de la Programación de Operaciones en la Industria y de la importancia de la misma. El Objeto del Proyecto también se recogerá en este apartado.

1.1. Contexto del Proyecto

Como definen Joaquín Bautista Valhondo y Francisco Javier Llovera Sáez, (Bautista Valhondo, J; Llovera Sáez, 2014), la Organización de la Producción es la disciplina que se preocupa por la definición de las estructuras de los sistemas productivos, que incluyen personas, conocimiento, medios y normas, y define además el conjunto de operaciones conceptuales y materiales que se ejecutan para obtener, transformar o transportar productos. Por tanto, la Organización de la Producción se preocupa: por estudiar y poner en práctica el diseño de productos y procesos, por fijar la dirección al sistema productivo y planificar las operaciones para cumplir sus objetivos, y, también, por explotar y controlar los sistemas productivos.

Dentro de la Organización de la Producción, podemos encontrar la Programación de la Producción (*Manufacturing Scheduling*) y el Control de la Producción (*Production Control*). La Programación de la Producción es un pilar fundamental para programar la producción de cualquier empresa o industria. Entendiéndose la programación de operaciones como la asignación de trabajos en un proceso productivo determinado, con la finalidad de optimizar una función objetivo. La secuenciación es una parte fundamental en los sistemas de producción y de fabricación de muchos de los sectores industriales. Este proyecto se centrará en un caso particular dentro de la secuenciación, en el que se buscará la minimización del tiempo de terminación de un trabajo específico, cumpliendo a su vez una serie de restricciones.

La programación de la producción no se entiende sin una planificación de transporte, una planificación de necesidades de material, la gestión de la demanda y un largo etcétera. Por tanto, es una disciplina que busca la optimización del funcionamiento de la industria; pero que tiene muchos frentes correlacionados que tienen que estar bien integrados para que funcione correctamente.

En primer lugar tiene que quedar claro qué se entiende por máquinas y por trabajos. Una máquina es un recurso productivo con capacidad para realizar operación de transformación o transporte de material, como por ejemplo un horno que realice un tratamiento térmico o un grupo de operarios que realicen una operación de montaje. Por otro lado, se entiende por trabajo un producto que es objeto de una operación en alguna de las máquinas de la fábrica, como por ejemplo el torneado de una pieza de acero al carbono.

En cualquier caso, un proceso industrial consiste en la toma de un número elevado de decisiones a lo largo del tiempo para asegurar la entrega al cliente de los productos con la máxima calidad, mínimo coste, y mínimo tiempo de entrega. Para alcanzar estos objetivos es clave una correcta programación de operaciones en la empresa. Por tanto, la programación de operaciones se puede decir que no solo tiene impacto sobre la productividad de la empresa, sino también sobre el beneficio económico o la satisfacción de los clientes.

1.2. Objeto del Proyecto

La programación se entiende como un proceso de toma de decisiones que tiene como fin la optimización de uno o más objetivos, (Pinedo, 1995). El objetivo del presente proyecto consiste en analizar un problema de secuenciación de taller de flujo regular de permutación, (*permutation flowshop*), con dos conjuntos de trabajos y objetivo makespan, también denominado longitud de programación. Un problema de programación de la producción consiste en programar un número de trabajos en un determinado número de máquinas con objeto de optimizar un determinado criterio fijado previamente. En este proyecto el objetivo a optimizar será el makespan de un conjunto, que se puede describir como el tiempo de finalización de todos los trabajos, es decir, se intentará minimizar el tiempo en el que termina el último trabajo en la última máquina. También se tendrá otro conjunto de trabajos del mismo tamaño que incorporará una restricción al problema. Cabe añadir que en los entornos *flowshop* o taller de flujo regular las máquinas se encuentran dispuestas en serie. Además, los trabajos pasan por cada una de las máquinas del sistema usando la misma ruta. Esto significa que todos los trabajos pasan por todas las máquinas en el mismo orden.

El problema en el que se centra este proyecto se basa en el estudio de programación de la producción de dos conjuntos de trabajos, grupo A y grupo B. Se ha optado porque ambos conjuntos tengan idéntico número de trabajos. Estos grupos de trabajos tienen respectivamente n_A y n_B, (n_A=n_B), trabajos, siendo la suma total de estos trabajos $n=n_A+n_B$. Estos dos conjuntos de trabajos se ejecutarán dentro del mismo entorno, en este caso, un entorno *flowshop* de m máquinas.

La notación para describir los diferentes tipos de entornos, de objetivos y de restricciones se presentará en el siguiente capítulo, donde se explica la Notación de Graham, (Graham et al., 1979). Para describir un problema de programación de manera completa se deben definir tres parámetros diferentes: el sistema o entorno a estudiar (α), las restricciones del sistema (β) y la función objetivo a optimizar (γ). Según la notación de Graham quedaría de la siguiente manera: $\alpha \mid \beta \mid \gamma$.

El sistema con el que se trabajará será en serie o *flowshop* de permutación de m máquinas, por tanto, $\alpha = Fm$. Como restricciones o condicionantes del problema solo aplicará la condición de permutación, $\beta = prmu$. Para que esto sea posible se harán una serie de suposiciones iniciales con respecto a las restricciones en un sistema real: El tiempo transcurrido al pasar de una máquina a otra es considerado despreciable, todos los trabajos están disponibles al principio del horizonte de programación, el tiempo de setup se supone incluido en el tiempo de proceso de cada trabajo, ninguna máquina puede realizar más de una operación simultáneamente y finalmente el buffer entre máquinas se supone infinito.

En este problema el objetivo no será generalizado para todos los trabajos. Como se tienen dos conjuntos de las mismas dimensiones se hará diferencia entre ellos, para darle otra visión un poco más compleja al problema. El objetivo a optimizar es el makespan únicamente del conjunto A, C^A_{max}, (tiempo de terminación del último trabajo del grupo A en la última máquina); sujeto a que el makespan del conjunto B, C^B_{max}, (tiempo de terminación del último trabajo del grupo B en la última máquina) no supere un valor límite, épsilon (ε), establecido con anterioridad. Este tipo de restricción de épsilon es denominada como *ε-constraint*. Por tanto, la función objetivo de estudio corresponde a $\gamma = \varepsilon(C^A_{max}, C^B_{max})$. También se puede expresar el criterio de optimización de la siguiente manera:

$$Min\ C^A_{max}$$
$$s.a.\ C^B_{max} \leq \varepsilon$$

El problema de estudio también se puede expresar con la siguiente notación de Graham:

$$Fm \mid prmu \mid \varepsilon(C^A_{max}, C^B_{max})$$

Finalmente se realizará una comparación de los resultados obtenidos en función de una serie de parámetros fijados, que se explicarán a lo largo del Proyecto. Se realizará un análisis de los datos obtenidos para extraer conclusiones. Para poder analizar y dar una solución aproximada para este problema se construirán una serie de

algoritmos y adaptaciones de heurísticas ya conocidas. Para una mejor explicación de los algoritmos se incluirán también los pseudocódigos donde se expresan los distintos pasos que deben realizar los programas de la forma más detallada posible, a la vez que compacta. Para todos los códigos de programación se ha utilizado lenguaje C++. Estos códigos se adjuntarán en la sección de Anexos de este Proyecto.

1.3. Estructura del documento

El documento está desglosarlo en diferentes epígrafes:

En el primer capítulo se ha realizado una contextualización del mundo de la programación de operaciones y de sus diferentes aplicaciones en la Industria. Tras esta contextualización se explica el Objetivo del Proyecto: Resolver un problema de programación de operaciones con dos agentes y objetivo makespan, basándonos en la heurística NEH, (Nawaz et al., 1983).

En el segundo capítulo se hace una descripción teórica de la secuenciación. Se explican los conceptos básicos necesarios para la comprensión del Proyecto, así como la notación que se utilizará a lo largo del mismo.

En el tercer capítulo se explicarán las instancias en las que se basará la resolución del Proyecto; y también se describirán las modificaciones que deben ser realizadas a dichas instancias para poder utilizarlas en la resolución del problema del Proyecto.

En el cuarto capítulo se presentarán las tres adaptaciones que se proponen en este Proyecto, con el fin de realizar una experimentación computacional para analizar y comparar sus resultados.

Por último, se presentan las conclusiones del Proyecto; donde se recoge una recapitulación de las ideas más relevantes que se han extraído.

2 INTRODUCCIÓN A LA SECUENCIACIÓN

Como recoge M.L. Pinedo en su libro *Scheduling: Theory, Algorithms, and Systems*, (Pinedo, 2012), la secuenciación es una forma de toma de decisiones que juega un papel fundamental en la industria. En un entorno tan competitivo como el actual, una secuenciación y programación eficiente se ha vuelto crucial para sobrevivir en el mercado. La secuenciación comenzó a tomar peso a principios de este siglo con trabajos como el de Henry Gantt y otros pioneros. Por tanto, la secuenciación, conocida como *Scheduling*, se puede definir como la asignación de trabajos en un proceso productivo determinado, con la finalidad de optimizar una función objetivo.

La resolución de los problemas de secuenciación dependen del entorno y de sus restricciones. Por tanto, estos problemas dependen de las características de las máquinas y su distribución, de las restricciones de los trabajos y del objetivo que se quiere optimizar. La notación para describir los diferentes tipos de sistemas se presentará en el siguiente subcapítulos 2.1., donde se explica la Notación de Graham, (Graham et al., 1979). Por tanto, la forma más básica de secuenciación consiste en la asignación de trabajos en diferentes máquinas con un determinado tiempo de proceso.

Cuando hablamos de problemas de secuenciación en la industria, el objetivo más importante es mejorar la productividad. Sin embargo, puede haber infinidad de objetivos a optimizar, como por ejemplo: reducir el tiempo ocioso de las máquinas, minimizar los setup de las mismas o minimizar el número de trabajos terminados después de su fecha de entrega porque esto significa clientes insatisfechos. Como apuntan José M. Framiñan y la tutora de este proyecto, Paz Pérez González, (Perez-Gonzalez & Framinan, 2014), la mayor parte de la investigación sobre programación clásica asume que los objetivos que se persiguen son comunes a todos los trabajos a realizar. Sin embargo, muchas de las aplicaciones de la vida real se pueden modelar considerando diferentes conjuntos de trabajos cada uno con sus propios objetivos. Como se puede observar la cantidad de tipos de problemas que se pueden estudiar y analizar es muy amplia.

Los problemas de secuenciación no solo se pueden dividir dependiendo de sus características sino también de su complejidad. Según la complejidad de los problemas se pueden clasificar en dos clases: Modelos con algoritmos polinomiales (P), o modelos con algoritmos no polinomiales (NP-hard). La mayoría de los problemas son NP-hard. Encontrar una solución óptima para una instancia real (NP-hard) en un tiempo razonable es poco probable, por lo que se buscan soluciones aproximadas o heurísticas.

Pero los problemas de secuenciación no solo tienen esta dificultad intrínseca, sino que existen muchos problemas relacionados con su implantación en las empresas. Sin embargo Kenneth Mckay, M.L. Pinedo y Scott Webster, (Mckay et al., 2002), exponen que aunque todavía no hay mucho impacto de la investigación de programación en las técnicas de fabricación, en los últimos años se está apreciando un crecimiento exponencial en el desarrollo e implantación de sistemas de programación en la industria. Una correcta implantación depende en gran medida de los datos tomados previamente para usarlos de datos de partida. A parte de la veracidad de los datos también hay que tener en cuenta el tamaño de la muestra y si los parámetros de la misma son realmente los idóneos para sacar conclusiones del problema. También es importante la adecuación del modelo teórico al problema real, (engineering problem: perfectly solving the wrong problem); y por supuesto puede ser un error muy grave utilizar métodos estadísticos inapropiados para analizar los resultados.

Este proyecto solo se centrará en la investigación y análisis de datos de una serie de supuestos, en un escenario ficticio, dejando a un lado la implantación de cualquier sistema de programación en una industria real. Pero para poder analizar y dar una solución aproximada para este problema se construirán una serie de algoritmos y adaptaciones de heurísticas ya conocidas.

Para una mejor explicación de los algoritmos se incluirán también los pseudocódigos donde se plasman los distintos pasos que deben realizar los programas de la forma más detallada posible, a la vez que compacta. Para todos ellos se utilizará una misma terminología que se expondrá a continuación. Sin embargo, algunos de los pseudocódigos que se presentarán a lo largo de este proyecto serán más específicos y tendrán sus propias variables por lo que contarán con un anexo a la terminología general que es la que se expone a continuación.

Terminología conjunta que se utilizará en los diferentes pseudocódigos:

- Número de máquinas (m), del entorno. Conjunto de máquinas $M = \{1, \dots, m\}$. Índices para las máquinas $i \in M$.
- Número de trabajos (n), a procesar. Conjunto de trabajos $N = \{1, \dots, n\}$. Índices para los trabajos $j \in N$.
- Tiempo de proceso (p_{ij}), es el tiempo que la máquina $i \in M$ está ocupada al procesar el trabajo $j \in N$.
- $Cmax_A$: Tiempo de terminación del último trabajo del grupo A en la última máquina del sistema.
- $Cmax_B$: Tiempo de terminación del último trabajo del grupo A en la última máquina del sistema.
- Π: Secuencia final encontrada por el algoritmo que minimiza el makespan de A, cumpliendo la restricción con respecto al makespan de B.
- S: Secuencia parcial que se va generando en las diferentes iteraciones de los algoritmos. Se le puede añadir un subíndice en el que se indique a que grupo de trabajos pertenece. Por ejemplo: S_A
- ε: Valor de épsilon para una instancia concreta. Se utilizará para imponer una restricción.
- AVG_j: Tiempo de procesamiento promedio del trabajo j,
- STD_j: Desviación estándar de los tiempos de procesamiento del trabajo
- LPT: Regla de prioridad Longest Processing Time. Los trabajos se secuencian en orden decreciente de tiempo de proceso.
- C_{max} El makespan es el intervalo de tiempo entre el inicio del procesamiento del primer trabajo y el tiempo de finalización del último. $C_{max} = \max_{1 \le j \le n} C_j$. También pueden acompañarse de subíndices o superíndices para especificar que se trata del makespan únicamente de un conjunto de trabajos: C^A_{max} y C^B_{max}

Aunque no van a ser necesarios para la el desarrollo de este proyecto, se incluirán a continuación una serie de conceptos también muy relevantes en la programación de operaciones.

- Ruta de proceso (R_j), vector que indica el orden en el que el trabajos j va a ser procesado en las máquinas.
- Fecha de llegada (r_j), instante de disponibilidad a partir del cual el trabajo $j \in N$ puede empezar a ser procesado.
- Fecha de entrega (d_j), instante de tiempo en el que el trabajos $j \in N$ debe estar terminado.
- Peso (w_j), prioridad del trabajos $j \in N$, respecto al resto de trabajos.

2.1. Notación de Graham

Para definir la programación de la producción a lo largo del proyecto se utilizará la notación de Graham, (Graham et al., 1979). Esta se basa en la siguiente expresión: α | β | γ. Se pueden definir distintos problemas de programación de la producción.

Existe un amplio espectro de características que pueden asociarse a los trabajos y al modo de procesamiento dentro del sistema.

Alpha (α), este campo define las características de las máquinas. Por una parte especifica el número de máquinas con las que se trabajará y por otro lado la distribución de las mismas. Solo constará de una entrada.

- Single Machine (*1*): También definida como Máquina única, es el entorno de máquinas más sencillo, consta de una sola máquina en la que se procesan todos los trabajos.

Existen también los entornos de máquinas dispuestas en paralelo. Los entornos de máquinas paralelas se pueden clasificar en diferentes tipos a su vez.

- Indentical Parallel Machine (*Pm*): Son máquinas idénticas dispuestas en paralelo y el tiempo de proceso de cada trabajo no depende de la máquina.
- Uniform Parallel Machine (*Qm*): Son máquinas dispuestas en paralelo pero que tienen diferentes velocidades que no dependen de los trabajos. v_i; Los tiempos de proceso de cada trabajo j en cada máquina i se pueden entender con la siguiente expresión $p_{ij} = p_j / v_i$.
- Unrelated Parallel Machine (*Rm*): Caso más general de máquinas paralelas, son máquinas diferentes. El tiempo de proceso de cada trabajo depende de la máquina a la que sea asignado p_{ij}.

Los entornos tipo taller se pueden subdividir. También entendidos como conjuntos de máquinas en serie.

- Flowshop (*Fm*): Máquinas en serie, todos los trabajos j tienen la misma ruta. Es decir, cada trabajo j es procesado en cada una de las máquinas en el mismo orden.
- Jobshop (*Jm*): Al igual que en el entorno *flowshop* se considera que todas las máquinas tienen funciones diferentes. La diferencia con el anterior es que cada trabajo tiene una ruta diferente, que tiene que ser establecida de antemano.
- Openshop (*Om*): También denominado taller abierto, es el más general de los entornos tipo taller. Los trabajos no tienen una ruta predeterminada, es decir, que no tienen restricciones en cuanto al orden de procesamiento en las máquinas.

Beta (*β*), este campo define las características de la programación del problema, es decir, indica las restricciones que tenga el problema. Puede constar de una sola entrada o de varias. Si las restricciones se aplican únicamente a un conjunto, se indicará con un superíndice. Algunas de las restricciones más usuales son las siguientes.

- Las relacionadas con las fechas de llegada (r_j), el trabajo j no puede ser procesado hasta esa fecha, y las fechas de entrega (d_j), el trabajo debe programarse para que sea entregado antes de la fecha límite marcada.
- La condición de permutación (*prmu*), todos los trabajos siguen la misma ruta, pero al pasar de una máquina a otra los trabajos no pueden adelantarse y alterar la secuencia.
- Precedencia (*prec*), esta condición puede ser aplicada a máquinas o a trabajos. Por ejemplo en el caso de las máquinas, está condición puede indicar que todos los trabajos deben ser procesados en la máquina i antes de pasar a la máquina j.
- No-wait (*no-wait*), sin interrupciones. Esta restricción marca que los trabajos deben de ser procesados sin ninguna interrupción, desde que comienza su procesamiento en la primera máquina hasta que finaliza en la última.

Gamma (γ), este campo indica cual es la función objetivo a optimizar en el problema. Normalmente se trata de un único objetivo, pero pueden ser varios también. Algunas de las funciones objetivo más usuales se presentan a continuación.

- Makespan (C_{max}), se define como el tiempo de terminación del último trabajo. $C_{max} = \max_{1 \le j \le n} C_j$,

donde C_j son los tiempos de terminación de cada trabajo.

- Total Completion (*SumCj*), que es la suma de los tiempos de finalización de todos los trabajos. $SumCj = \sum_{j=1}^{n} C_j$.
- Maximun Lateness (*max Lj*), o máximo retraso. Siendo Lj el retraso de cada trabajo j, es decir, la diferencia entre la fecha de entrega prevista y la fecha en la que termina el trabajo.
- Podemos encontrar muchos más como el Máximum Tardiness (*max Tj*), Maximum Earliness (*max Ej*), Número de trabajos que van tarde ($\sum U_j$), etc. Todos estos objetivos también pueden plantearse ponderados, es decir, multiplicando cada uno por su peso asignado w_J.

2.2. Secuenciación de varios conjuntos de trabajos

Como se puede ver en (Pinedo, 1995) los problemas de secuenciación también se pueden definir cómo la asignación de trabajos en un proceso de fabricación que tiene como fin la optimización de uno o más objetivos. Sin embargo, en los problemas de secuenciación que se dan en situaciones reales no solo tienen más de un objetivo, si no que también, pueden considerarse más de un conjunto de trabajos, cada uno con sus propios objetivos. Hoogeveen, (Hoogeveen, 2005), ya mencionó los problemas multicriterio con dos o más conjuntos de trabajos. Este tipo de problemas con varios conjuntos de trabajos han sido denominados en la literatura como *multi-agent scheduling problems, interfering jobs y mixed-criteria problems,* (Perez-Gonzalez & Framinan, 2014). Además en este artículo también se estudia la terminología y la notación empleada para este tipo de problemas. Existe un amplio rango en cuento a nomenclatura y objetivos en este tipo de problemas, por tanto, Pérez-González y Framiñan proponen como término para unificarlos todos: *multi-agent scheduling problem* (MASP), ya que es el más utilizado. Las aplicaciones de este tipo de problemas en la vida real abarcan un gran número de campos tales como la secuenciación de las cadenas de suministro, la reprogramación o actualización de un sistema de producción existente en respuesta a imprevistos o cambios, la secuenciación de problemas relacionados con las telecomunicaciones o la secuenciación del mantenimiento preventivo de manera simultánea al sistema de producción. Por todas estas aplicaciones este tipo de problemas han sido objeto de interés por parte de investigadores y profesionales en los últimos años.

Considerando dos o más conjuntos de trabajos, que compiten y usan los mismos recursos o máquinas, entendemos que cada uno de los conjuntos de trabajos tiene un objetivo, que puede o no, ser el mismo. O bien que los objetivos de algunos conjuntos deban optimizarse mientras otros estén sujetos a diferentes restricciones. (Perez-Gonzalez & Framinan, 2014). El problema a tratar en este Proyecto es del segundo tipo, optimizar el objetivo del primer conjunto de trabajos sujeto a que el segundo conjunto satisfaga una restricción. Como se recoge en (Perez-Gonzalez & Framinan, 2014) existen diferentes enfoques para la minimización de objetivos para dos conjuntos de trabajos. Entre ellos se encuentra el enfoque de restricción de épsilon (ε-*constraint*) que consiste en minimizar el objetivo de un conjunto de trabajos sujeto a que el objetivo del segundo conjunto de trabajos sea menor o igual que épsilon. Una instancia de un problema de restricción de épsilon puede no tener soluciones factibles si épsilon tiene un valor reducido. Entendiéndose como factible si al menos una de las soluciones en factible y satisfice las condiciones del problema.

3 INSTANCIAS DEL PROBLEMA

En el primer apartado 1.2. se ha desarrollado el Objetivo del proyecto y se ha definido el problema a resolver. Para poder hacer la experimentación computacional de este problema se utilizarán una serie de instancias para poder conseguir resultados para su posterior análisis. Las instancias son archivos donde se almacenan las características básicas de un sistema, es decir, el número de máquinas, el número de trabajos y los tiempos de proceso de los mismos en cada máquina. En este apartado se explicarán las instancias escogidas y las modificaciones que hay que aplicarles para poder usarlas con el objetivo y las restricciones del problema actual.

3.1 Instancias de Taillard

Para la resolución de este problema se hará uso de las instancias propuestas por Taillard, (Taillard, 1993). En el artículo citado, Taillard propone 260 instancias para problemas de secuenciación; los tiempos de proceso de cada trabajo en cada máquina son generados de manera aleatoria para cada instancia siguiendo una distribución uniforme entre 1 y 99. Los tiempos de proceso se obtienen de la siguiente manera para cada operación i de un trabajo j ($1 \leq i \leq m; 1 \leq j \leq n$):

Para j=1 hasta n:

Para i=1 hasta m:

$p_{ij} := U[1,99];$

Ilustración 3-1: Código de Programación de una distribución uniforme U[1,99].

De las 260 instancias, en este proyecto se utilizarán un total de 120 correspondientes a las diseñadas para problemas con entorno de tipo *flowshop*. Las dimensiones de estas instancias corresponden a dimensiones reales de problemas en la industria. Estas instancias tienen un rango desde 5 hasta 20 máquinas; y para los trabajos, el número mínimo de trabajos es de 20 y el número máximo de trabajos que podemos encontrar en una instancia es 500. La distribución de máquinas y de trabajos que aparece en la Tabla 3.1. corresponden a las 120 instancias que se utilizarán en este proyecto.

Estas instancias proporcionan los datos de un problema básico de secuenciación: número de máquinas, número de trabajos total y también el tiempo de proceso de cada trabajo en cada máquina. Dentro de las características de las instancias no aparecen ni los trabajos pertenecientes a cada grupo ni el valor de épsilon para la restricción de ε-*constraint* de este problema.

Instancias	Número de máquinas	Número de trabajos
1-10	5	20
11-20	10	20
21-30	20	20
31-40	5	50
41-50	10	50
51-60	20	50
61-70	5	100
71-80	10	100
81-90	20	100
91-100	10	200
101-110	20	200
111-120	20	500

Tabla 3.1: Distribución de máquinas y trabajos en las Instancias de Taillard

3.2 Generación del parámetro épsilon

Teniendo en cuenta que se ha definido el criterio de optimización del problema a tratar como:

$$\gamma = \varepsilon(C^A_{max}, C^B_{max})$$

El primer valor necesario para comenzar a modelar el problema es el parámetro épsilon, ε. Observando las características de las instancias que se van a utilizar, Instancias de Taillard, se observa que el valor de épsilon no es proporcionado por las mismas. Por tanto, para poder hacer la experimentación con las instancias lo primero es obtener unos valores apropiados para épsilon. La restricción épsilon-constrait, $C^B_{max} \leq \varepsilon$, impone un límite superior al valor C^B_{max}, el makespan del conjunto B de trabajos. Cuanto más holgada sea la restricción, o lo que es lo mismo, cuanto mayor sea el valor de ε, habrá un mayor número de secuencias factibles y el problema podría reducirse a un problema tipo: $Fm \mid prmu \mid C^A_{max}$.

Para poder generar un valor de épsilon admisible se ha tomado como referencia el algoritmo NEH, (Nawaz et al., 1983), que se definirá a continuación. Sin embargo, esta heurística será adaptada para poder utilizarla con dos conjuntos de trabajos. Con la NEH adaptada a dos conjuntos de trabajos se generarán valores de épsilon de forma que sean dependientes de los demás datos de la instancia: número de máquinas, número de trabajos y los tiempos de proceso. Por otra parte, se estudiarán dos adaptaciones de la heurística NEH, con el fin de escoger los valores de épsilon más adecuados para las instancias.

3.2.1. Algoritmo NEH

La heurística constructiva NEH fue desarrollada por Nawaz, Enscore Jr. y Ham, (Nawaz et al., 1983). Esta heurística busca minimizar el instante máximo de finalización del conjunto de los trabajos (makespan), en problemas flow shop ha demostrado ser un buen método de resolución.

En este procedimiento pueden diferenciarse dos fases: en la primera se ordenan los trabajos de mayor a menor tiempo total de proceso (regla LPT); y la segunda fase o fase de inserción, se construyen las secuencias parciales buscando minimizar el makespan parcial. Los pasos de la segunda fase son los siguientes:

1. Con los trabajos ordenados según la regla LPT, se evalúan los dos primeros trabajos de esta secuencia, intercambiando sus posiciones para evaluar el makespan.
2. La secuencia parcial de inicio será la que menor makespan tenga de las dos evaluadas en el paso anterior.
3. De forma iterativa se van incluyendo el resto de trabajos (en el orden establecido por la regla LPT) en las diferentes posiciones posibles de la secuencia parcial generada, evaluando su makespan parcial, y seleccionando aquella secuencia que minimice el makespan. Una vez colocado un trabajo, la posición de este con respecto al resto de trabajos ya secuenciados no variará en las siguientes iteraciones.

Nota 1: En caso de que dos secuencias parciales en las que se ha introducido un mismo trabajo pero en distintas posiciones, tuviera el mismo valor para el makespan; el algoritmo selecciona aquella que se haya generado en último lugar. Esta regla de desempate se denomina FS.

Nota 2: En la primera fase, en caso de que los tiempos de proceso totales coincidieran en dos trabajos, para construir la secuencia según la regla LPT, se ordenarán en función del índice de trabajo, colocando en primer lugar aquel que tenga menor índice.

Para completar la explicación de la heurística NEH se ejemplificará con un entorno sencillo que consta de cuatro trabajos y cuatro máquinas. La Tabla 3.2. muestra los tiempos de proceso de cada trabajo en cada máquina.

Máquinas\Trabajos	*J1*	*J2*	*J3*	*J4*
M1	3	3	2	1
M2	2	3	4	5
M3	4	2	3	4
M4	5	3	3	3

Tabla 3.2: Matriz de tiempos de proceso de los trabajos J_j en la máquina M_i

Primera fase: Se hace un sumatorio de los tiempos de proceso de cada trabajo en todas las máquinas del entorno. En el caso estudiado asociado a la Tabla 3.2:

- J_1: 3 + 2 + 4 + 5 = 14
- J_2: 3 + 3 + 3 + 2 = 11
- J_3: 2 + 4 + 3 + 3 = 12
- J_4: 1 + 5 + 4 + 3 = 13

Con los tiempos de proceso totales se procede a secuenciarlos según la regla LPT, (Largest Processing Time). En el entorno actual sería: (J_1, J_4, J_3, J_2).

Segunda fase o fase de inserción: Se seleccionan los dos primeros trabajos de la secuencia anteriormente generada y se evalúan las dos secuencias posibles: (J_1, J_4) o (J_4, J_1).

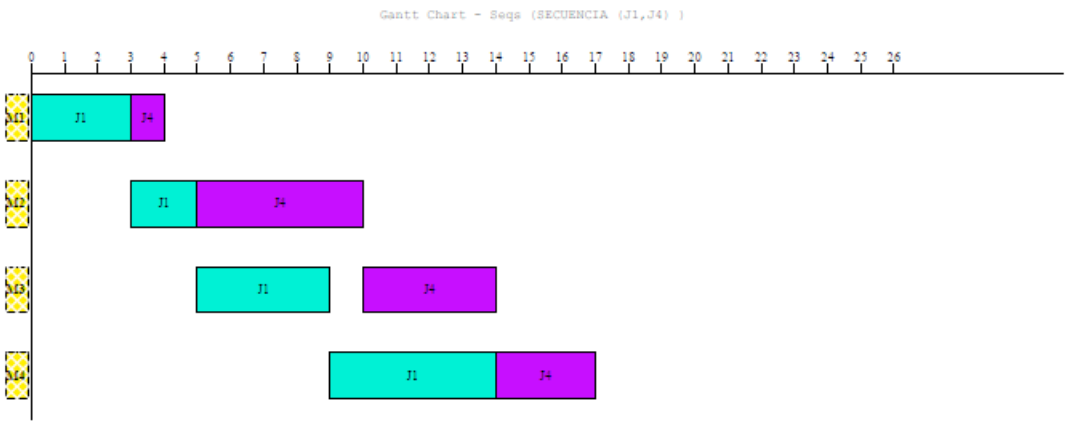

Figura 3.1: Diagrama de Gantt para la secuencia (J_1, J_4), con makespan 17

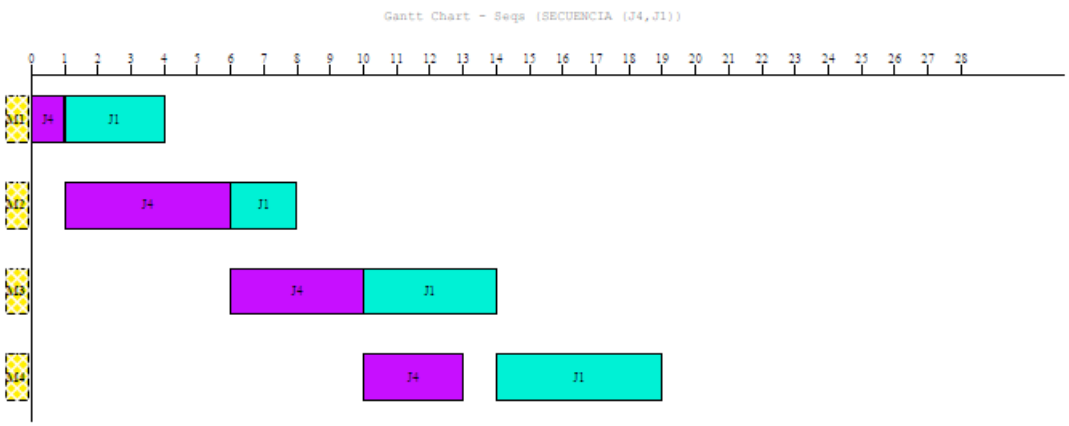

Figura 3.2: Diagrama de Gantt para la secuencia (J_4, J_1), con makespan 19

Para este primer paso, se puede ver en las Figura 3.1 y Figura 3.2 que el makespan conseguido es respectivamente 17 y 19 unidades de tiempo. Por tanto, la secuencia parcial que minimiza el makespan es la (J_1, J_4). Esto indica que para las siguientes iteraciones el trabajo J_1 deberá ser ejecutado antes del trabajo J_4, aunque no necesariamente deberán ser consecutivos.

Tras haber seleccionado la secuencia parcial anterior, se inserta el siguiente trabajo de la secuencia LPT. Para el entorno planteado en este ejemplo es el trabajo J3. Las siguientes Figura 3.3, Figura 3.4 y Figura 3.5, muestran los Diagramas de Gantt de las posibles combinaciones de los trabajos. La secuencia que minimiza el makespan es la última generada: (J_1, J_4,J_3). Para la cual el makespan tiene un valor de 20 unidades de tiempo.

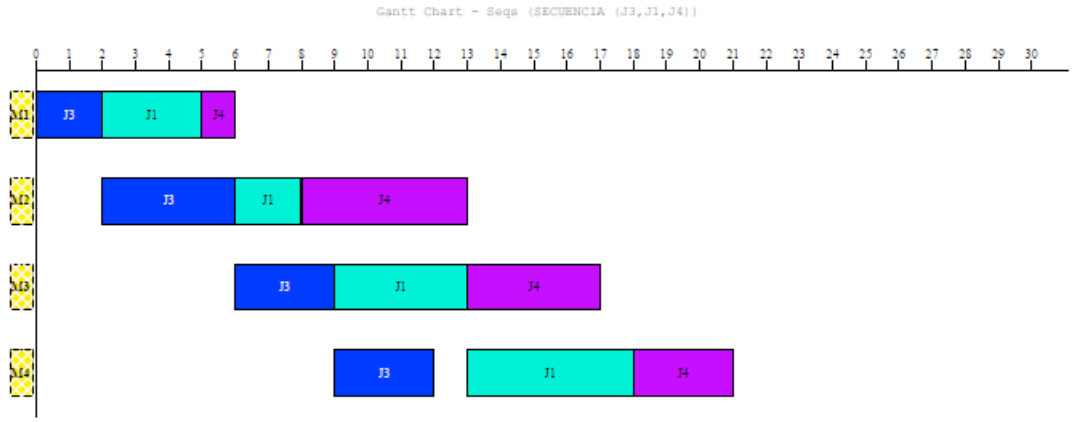

Figura 3.3: Diagrama de Gantt para la secuencia (J_3,J_1, J_4), con makespan 21

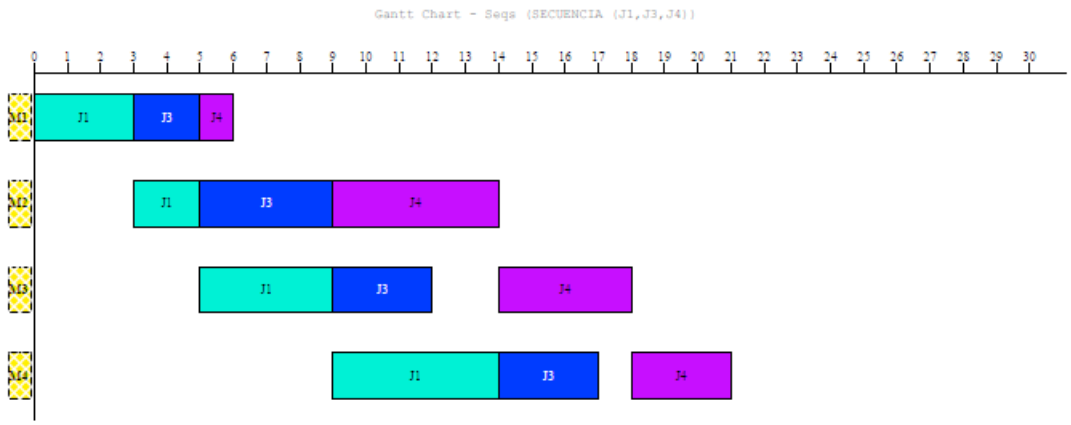

Figura 3.4: Diagrama de Gantt para la secuencia (J_1,J_3, J_4), con makespan 21

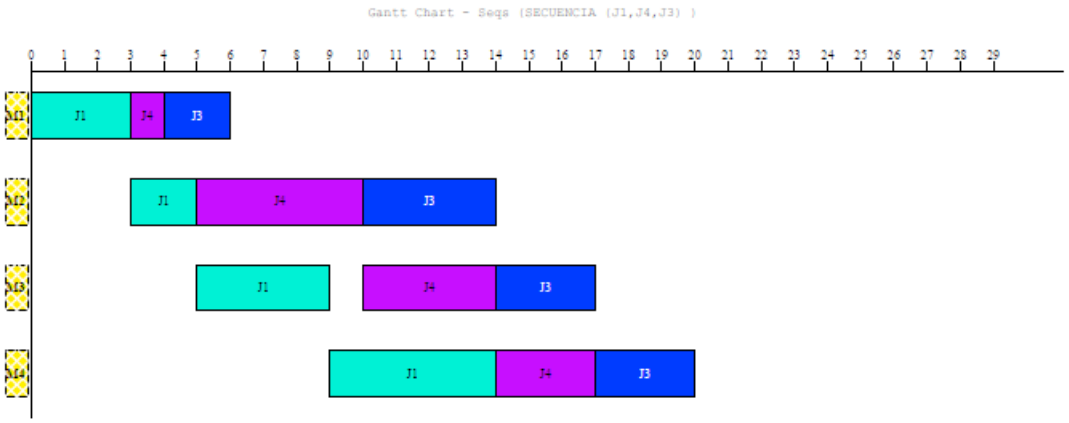

Figura 3.5: Diagrama de Gantt para la secuencia (J_1,J_4, J_3), con makespan 20

De nuevo se vuelve a hacer una iteración en el algoritmo para incluir el último trabajo de este entorno, en este caso: J_2, que es el que tiene menor tiempo de proceso. Las Figura 3.8, Figura 3.7, Figura 3.8 y Figura 3.8 muestran los Diagramas de Gantt de las posibles combinaciones de los cuatros trabajos. Sabiendo que la secuencia parcial anterior era (J_1,J_4, J_3) habrá que introducir el trabajo J_2 en todas las posibles posiciones de la secuencia.

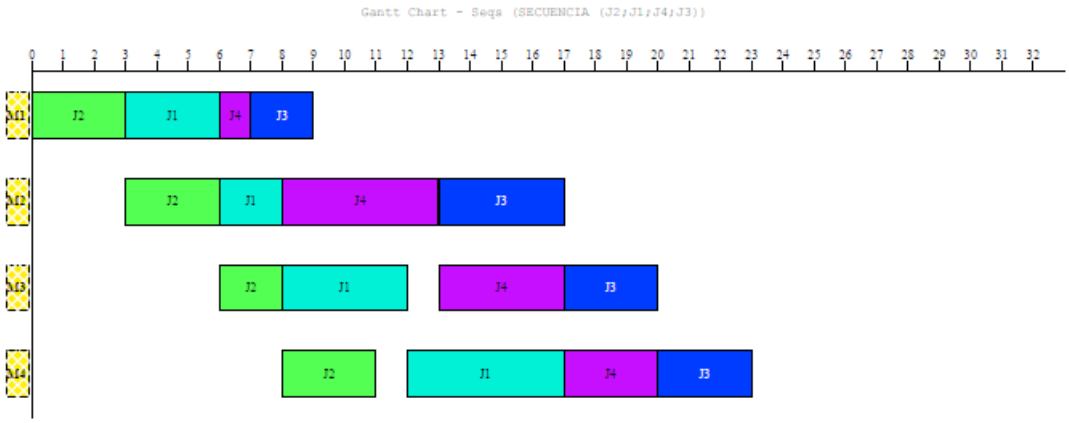

Figura 3.6: Diagrama de Gantt para la secuencia (J_2,J_1,J_4, J_3), con makespan 23

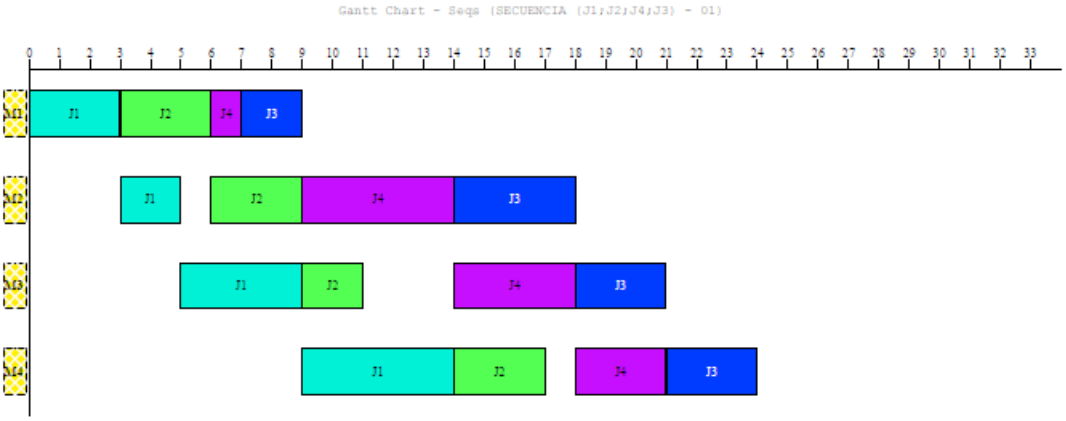

Figura 3.7: Diagrama de Gantt para la secuencia (J_1, J_2, J_4, J_3), con makespan 24

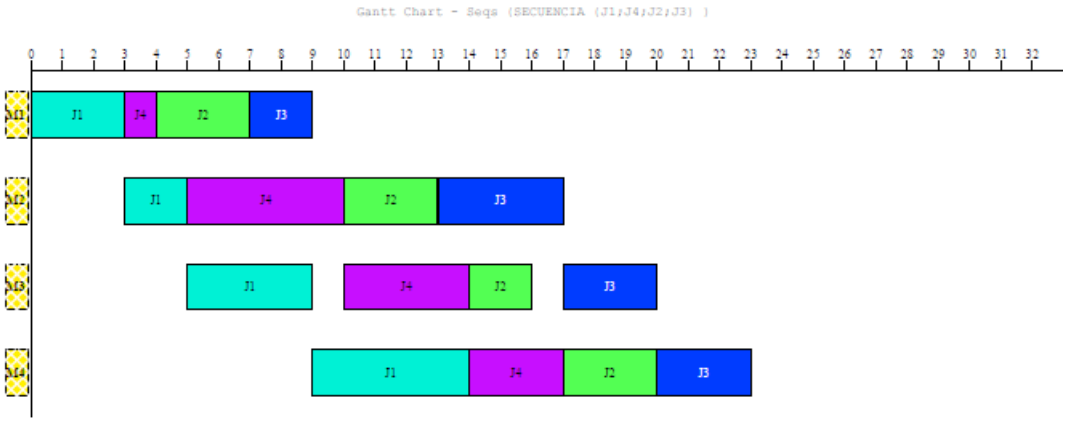

Figura 3.8: Diagrama de Gantt para la secuencia (J_1, J_4, J_2, J_3), con makespan 23

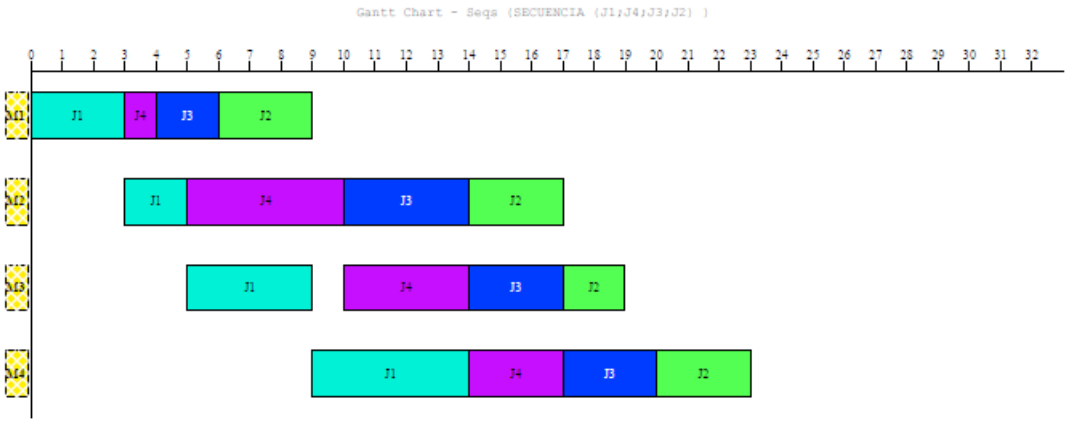

Figura 3.9: Diagrama de Gantt para la secuencia (J_1, J_4, J_3, J_2), con makespan 23

Como se puede comprobar, tres de las cuatro secuencias generadas dan un makespan del mismo valor: 23 unidades de tiempo. Se ha seleccionado este ejemplo para mostrar cómo se implementa la Nota 1 explicada anteriormente. En la cual se selecciona la última secuencia generada por el algoritmo. Por tanto, la secuencia dada por el algoritmo NEH para encontrar el mínimo makespan sería (J_1, J_4, J_3, J_2).

3.2.2. Adaptación algoritmo NEH para generar el valor de épsilon

A continuación se recoge el pseudocódigo de la heurística NEH adaptada para dos conjuntos de trabajos A y B. Ambos grupos constan de igual número de trabajos, es decir, $n = n_A + n_B$, siendo $n_A = n_B$. Por tanto para ordenar los trabajos en las máquinas primero se aplicará la heurística NEH para el grupo A, NEH^A ($j=1...n/2$) y posteriormente se aplicará la heurística NEH^B para el grupo B ($j=(n/2)+1...n$). Una vez esté definida la secuencia que optimiza según la heurística NEH los trabajos de grupo A, se aplicará NEH^B. NEH^B irá introduciendo los trabajos en las secuencias parciales solo en las posición posteriores a $n/2$, es decir, que no se alterará la secuencia parcial establecida para el grupo A. Por tanto, se obtendrá una secuencia con todos los trabajos de A ordenados según NEH^A ocupando las primeras posiciones y con todos los trabajos de B ordenados según NEH^B ocupando la segunda mitad de las posiciones de la secuencia. Una vez construida esta adaptación de la heurística se utilizará para generar valores de épsilon.

En el pseudocódigo se expresan los distintos pasos que debe realizar el programa de la forma más detallada posible. El código de programación elaborado para realizar las estimaciones de valores de épsilon con NEH de este apartado, se encuentra recogido en la sección final de Anexos de este documento, concretamente en el Anexo A.

INPUT: datos de la instancia (número de máquinas, número de trabajos y matriz de tiempos de proceso)

OUTPUT: Π, Cmax

Begin

$P_j = \sum_{i=1}^{m} P_{ij}$;

Dividir en dos grupos: P_A donde se almacenen P_j con $J = \{1, \dots, n/2\}$ y P_B donde se almacenan P_j con $J = \{\frac{n}{2} + 1, \dots, n\}$;

Π = ∅ , S_A= {1,…, n/2} verificando $P_{A\,1} \geq P_{A\,2} \geq \cdots \geq P_{A\,n/2}$;

Π que sea la mejor permutación entre [1,2] y [2,1];

Para k=3 hasta n/2:

Insertar el trabajo k en la posición de Π que verifique el menor valor para el makespan;

S_B= {1,…, n/2} verificando $P_{B\,1} \geq P_{B\,2} \geq \cdots \geq P_{B\,n/2}$;

Para k= 1 hasta n/2:

Insertar el trabajo k en la posición de Π, siendo esta posición perteneciente a la segunda mitad de Π, de la posición $\frac{n}{2} + 1$ en adelante, que verifique el menor valor para el makespan;

Devuelve Π, Cmax

Figura 3.10. Adaptación algoritmo NEH para determinar el valor de épsilon

3.2.3. Adaptación algoritmo NEHFF para generar el valor de épsilon

En este apartado se define una adaptación para dos conjuntos de trabajos de la heurística constructiva NEHFF, (Fernández-Viagas & Framinan, 2014b), que se utilizará posteriormente para encontrar el valor de épsilon. Esta heurística tiene una serie de modificaciones con respecto al algoritmo NEH original. En el citado artículo se prueba que se obtienen estadísticamente significativamente mejores resultados que para los mecanismos de desempate ya existentes para problemas de programación con entorno *flowshop* y con objetivo makespan. Por ello se ha optado por probar también esta heurística para generar épsilon.

Para entender esta variante se utilizará la siguiente notación formada por tres campos (Fernández-Viagas et al., 2017): NEH(a $|b\ |c$). Donde los campos a, b y c son los que dan las diferentes variantes NEH:

- a: Secuencia inicial usada por NEH.
- b: Mecanismo de desempate implementado en NEH para seguir evolucionando las secuencias parciales.
- c: Este campo está asociado con la propiedad de reversibilidad del problema (Ribas et al, 2010). Establece que el makespan de una secuencia de permutación Π= ($\pi_1, \dots, \pi_n$). (en una instancia formada por n trabajos y m máquinas con tiempos de proceso asociados p_{ij}), es el mismo que el makespan de la secuencia inversa Π'= ($\pi_n, \dots, \pi_1$).

La heurística de referencia tomada para resolver el problema es: NEHFF, (Fernández-Viagas & Framinan, 2014b). Según la notación explicada anteriormente, (Fernández-Viagas et al., 2017), se expresa como:

$$\text{NEH}\ (AD|FF\ |d)$$

De aquí en adelante se seguirán utilizando la nomenclatura y la notación explicada en el capítulo anterior. Los campos del modelo son:

- AD: (Dong et al., 2008). Como es el primer campo es el criterio de clasificación de los trabajos para elegir la secuencia inicial que utilizará NEH. Esta regla de prioridad se basa en la siguiente hipótesis: a mayor desviación de los tiempos de procesamiento de un trabajo en cada máquina, mayor prioridad debería tener. En primer lugar se define $AVG_j\ y\ STD_j$.

  $$AVG_j = \frac{1}{m}\sum_{i=1}^{m} p_{ij}$$

  $$STD_j = \left[\frac{1}{m-1}\sum_{i=1}^{m}(p_{ij} - AVG_j)^2\right]^{\frac{1}{2}}$$

  Siendo AVG_j, el tiempo de procesamiento promedio del trabajo j, y STD_j la desviación estándar de los tiempos de procesamiento del trabajo j. Existen tres reglas de prioridad Avg, Dev y AvgDev, donde Avg significa que se ordenarán los trabajos según el órden descendente de AVG_j, que significaría hacerlo igual que en NEH. Dev significa ordenar los trabajos según el órden descendiente de STD_j, y AvgDev que significa que se ordenan los trabajos en órden descendiente de $AVG_j + STD_j$. Resultados (Dong, Huang y Chen, 2008) muestran que el NEH modificado con AvgDev rinde de forma más óptima. Por tanto para modelar el problema del proyecto utilizaremos esta última regla de prioridad: AvgDev.

- FF: (Fernández-Viagas & Framinan, 2014a). Como es el segundo campo es el mecanismo de desempate en NEH. Este mecanismo este relacionado con la minimización de los tiempos de inactividad total, es decir, que el criterio de desempate en el caso de que los makespan, C_{max}, de dos secuencias parciales sean iguales es aquel que tenga menor tiempo de inactividad total. Según Framinan, Leisten y Rajendran, (Framinan et al., 2003), la definición del tiempo de inactividad de una máquina no está definida de forma inequívoca y en la literatura se han utilizado varias formas diferentes. En este mecanismo se ha tomado la definición siguiente: En el tiempo de inactividad se considera los retrasos frontales (tiempos antes del primer trabajo) y el tiempo inactivo entre trabajos en cada máquina, pero se excluirán los retrasos posteriores (tiempo después del último trabajo en la máquina).

  Según esta definición se puede expresar el tiempo de inactividad de la máquina i con la siguiente ecuación, $it_i = C_{in} - \sum_{j=1}^{n} p_{ij}$ y el tiempo de inactividad total de la secuencia como $it = \sum_{i=1}^{m} it_i$.

  Sin embargo, (Fernández-Viagas & Framinan, 2014a), proponen utilizar una aproximación para calcular el tiempo de inactividad. Por tanto, al insertar un trabajo no programado r en todas las posiciones de la subsecuencia se calcula el makespan en función de las ecuaciones definidas por Taillard, (Taillard, 1990). Las cuales se muestran a continuación.

  Para una secuencia parcial de longitud *k-1*, antes de introducir el trabajo no programado, se puede definir e_{ij}, como el tiempo de finalización más temprano del trabajo j en la máquina i, antes de insertar el trabajo no programado.

  $$e_{ij} = max\{e_{i,j-1}, e_{i-1,j}\} + p_{ij};\ \ i = 1 \dots m;\ \ j = 1 \dots k-1;\ \ con\ e_{0j} = 0\ y\ e_{i0} = 0;$$

  De la misma manera se puede definir q_{ij}, que es el tiempo entre el tiempo de inicio del trabajo j en la máquina i, y el final de todas las operaciones que tengan que llevarse a cabo en la programación de los demás trabajos.

  $$q_{ij} = max\{q_{i,j+1}, q_{i+1,j}\} + p_{ij};\ \ i = m \dots 1;\ \ j = 1 \dots k-1;\ \ con\ q_{m+1,j} = 0\ y\ q_{ik} = 0;$$

  También se debe definir el tiempo de terminación de cada trabajo en cada máquina denotando la

posición del trabajo con la letra *l*. Siendo t_{ir}, el tiempo de procesamiento en la máquina i del trabajo r a insertar.

$$f_{il} = max\{e_{i,l-1}, f_{i-1,j}\} + t_{ir};\ \ i = 1 \dots m;\quad con\ f_{0,l} = 0;$$

Por tanto, el makespan, C_{max}, para la secuencia en la que se ha introducido el trabajo r en la posición l, se puede expresar con la siguiente ecuación:

$$C_{max}(\pi) = \max_{i=1\dots m} \{f_{il} + q_{il}\};$$

El mecanismo de desempate propuesto si dos secuencias parciales tienen un mismo makespan se basa en estimación del nuevo tiempo de inactividad indicado por $it'(l)$, para cada posición l para la cual se produce el empate.

$$it'(l) = \sum_{i=1}^{m} (f_{il} - e_{il} + p_{il} - t_{ir} + max\left\{f'_{i-1,l} - f_{i,l},\ 0\right\});$$

Siendo $f'_{il} = max\{f_{i,l}, f'_{i-1,l}\} + p_{il};\ \ i = 1 \dots m;\quad con\ f'_{0l} = 0;$ siendo p_{il} el tiempo de procesamiento del trabajo que estaba antes en la posición l.

- d: Este campo está asociado con la propiedad de reversibilidad. El valor *d* indica que se utilizará la secuencia de forma directa, y *di* se utiliza en este campo para determinar que se utilizará la secuencia inversa para evaluar el objetivo.

Tras describir la heurística original, el modelo debe ser adaptado para el problema actual, en el que existen dos conjuntos de trabajos, grupo A y grupo B.

En primer lugar para determinar la secuencia inicial de partida de NEH, se procederá a calcular AVG_j, el tiempo de procesamiento promedio del trabajo j, y STD_j la desviación estándar de los tiempos de procesamiento de j de todo el conjunto de trabajos $n = n_A + n_B$. Sin embargo, para calcular la secuencia inicial se ordenarán los trabajos dentro de cada grupo en función de la regla AvgDev de manera independiente. Por tanto se obtendrá una primera secuencia ordenada del conjunto A según la regla AD y por otro lado se obtendrá una segunda secuencia para el conjunto B ordenada según la regla AD.

Según la heurística NEH, una vez establecida la secuencia inicial se van evaluando las secuencias parciales, eligiendo aquella que tenga menor makespan. Sin embargo, ante la tesitura de que dos secuencias parciales dentro de una misma iteración tuvieran un makespan parcial de igual valor, la heurística marcaba que se eligiera aquella secuencia parcial generada en último lugar. Sin embargo aquí se aplicará el mecanismo de desempate FF (Fernández-Viagas & Framinan, 2014b). Al igual que la heurística NEH adaptada, la heurística NEHFF adaptada, primero se aplicará al conjunto de trabajos A y posteriormente al conjunto de trabajos B. Por tanto para ordenar los trabajos en las máquinas primero se aplicará la heurística NEHFF para el grupo A, $NEHFF^A$ ($j=1...n/2$) y posteriormente se aplicará la heurística $NEHFF^B$ para el grupo B ($j=(n/2)+1...n$). Una vez esté definida la secuencia parcial generada según la heurística $NEHFF^A$, se aplicará $NEHFF^B$. $NEHFF^B$ irá introduciendo las secuencias parciales de los trabajos solo en las posición posteriores a $n/2$, es decir, que no se alterará la secuencia parcial establecida para el grupo A. Por tanto, se obtendrá una secuencia con todos los trabajos de A ordenados según $NEHFF^A$ ocupando la primera mitad de las posiciones y con todos los trabajos de B, ordenados según $NEHFF^B$, ocupando la segunda mitad de las posiciones de la secuencia.

Con respecto al último campo *d*, su implantación es la misma para un solo grupo de trabajos como para dos grupos de trabajos.

En el pseudocódigo se expresan los distintos pasos que debe realizar la heurística adaptada NEHFF de la forma más detallada posible. El código de programación elaborado para realizar la heurística NEHFF adaptada se encuentra recogido en la sección final de Anexos de este documento, concretamente en el Anexo B.

INPUT: Datos de la instancia (número de máquinas, número de trabajos y matriz de tiempos de proceso)

OUTPUT: Π, Cmax, Tiempo de inactividad total para la secuencia Π

Begin

Para cada trabajo j (j=1…n) se obtiene:

$AVG_j = \frac{1}{m}\sum_{i=1}^{m} p_{ij}$;

$STD_j = \left[\frac{1}{m-1}\sum_{i=1}^{m}(p_{ij} - AVG_j)^2\right]^{\frac{1}{2}}$;

Y se suman ambos valores: Pj=AVG_j+STD_j;

Dividir en dos grupos: P_A donde se almacenen P_j con $J = \{1, \dots, n/2\}$ y P_B donde se almacenan P_j con $J = \{\frac{n}{2} + 1, \dots, n\}$;

Π = ∅ , S_A= {1,…, n/2} verificando $P_{A\,1} \geq P_{A\,2} \geq \cdots \geq P_{A\,n/2}$;

Π es asociada a [1,2] o [2,1], a la que tenga menor makespan, en caso de empate se seleccionará aquella con menor tiempo de inactividad, calculando: $it_i = C_{i2} - \sum_{j=1}^{2} p_{ij}$ y posteriormente $it = \sum_{i=1}^{m} it_i$;

Para k=3 hasta n/2:

Determinar los valores de $e_{ij}, f_{ij}\ y\ q_{ij}$, de la aceleración de taillard para la secuencia Π.

Insertar el trabajo k en todas las posiciones desde 1…k en la secuencia Π y evaluar el makespan.

bp= primera posición donde el makespan es mínimo.

tb= número de posiciones donde el makespan es mínimo

ptb= vector donde se guardan las posiciones donde el makespan es mínimo.

$itbp$= tiempo de inactividad correspondiente a bp, se fija en un valor relativamente grande.

If (tb>1 && k<n/2) **then**

Para l=1 hasta tb:

It''=0;

If (ptb[l]=k) **then:**

Para i=2 hasta m:

$It'' = It'' - e_{i,k-1} + f_{ik} - t_{ir}$;

Else:

$\boldsymbol{f'_{1,ptb[\,l]} = f_{1,ptb[l]} + p_{1,ptb[l]}}$;

Para i=2 hasta m:

$it'' = it'' + f_{i,ptb[l]} - e_{i,ptb[l]} + p_{i,ptb[l]} - t_{ir} + max\left\{f'_{i-1,ptb[l]} - f_{i,ptb[l]},\ 0\right\}$;

$$f'_{i,ptb[l]} = max\{f_{i,ptb[l]}, f'_{i-1,ptb[l]}\} + p_{i,ptb[l]};$$

If ($itbp < it''$**) then:**

bp=ptb[l];

itbp=it'';

Π será la secuencia obtenida al insertar el trabajo r en la posición bp, que verifica el menor makespan parcial.

S_B= {1,…, n/2} verificando $P_{B\,1} \geq P_{B\,2} \geq \cdots \geq P_{B\,n/2}$;

Para k= 1 hasta n/2:

Determinar los valores de e_{ij}, f_{ij} y q_{ij}, de la aceleración de taillard para la secuencia Π.

Insertar el trabajo k, en todas las posiciones posibles, siendo estas posiciones pertenecientes a la segunda mitad de Π, de la posición $\frac{n}{2} + 1$ en adelante, en la secuencia Π y evaluar el makespan.

bp= primera posición donde el makespan es mínimo.

tb= número de posiciones donde el makespan es mínimo

ptb= vector donde se guardan las posiciones donde el makespan es mínimo.

$itbp$= tiempo de inactividad correspondiente a bp, se fija en un valor relativamente grande.

If (tb>1 && k<n/2**) then**

Para l=1 hasta tb:

It''=0;

If (ptb[l]=k**) then:**

Para i=2 hasta m:

$$It'' = It'' - e_{i,k-1} + f_{ik} - t_{ir};$$

Else:

$$f'_{1,ptb[l]} = f_{1,ptb[l]} + p_{1,ptb[l]};$$

Para i=2 hasta m:

$$it'' = it'' + f_{i,ptb[l]} - e_{i,ptb[l]} + p_{i,ptb[l]} - t_{ir} + max\{f'_{i-1,ptb[l]} - f_{i,ptb[l]},\ 0\};$$

$$f'_{i,ptb[l]} = max\{f_{i,ptb[l]}, f'_{i-1,ptb[l]}\} + p_{i,ptb[l]};$$

If ($itbp < it''$) **then:**

bp=ptb[l];

itbp=it'';

Π será la secuencia obtenido al insertar el trabajo r en la posición bp, que verifica el menor makespan parcial.

Devuelve Π, Cmax

Figura 3.11: Adaptación algoritmo NEHFF para generar el valor de épsilon

3.2.4. Valores de épsilon

Con las dos adaptaciones de las heurísticas NEH y NEHFF para dos conjuntos de trabajos, expuestas en los subcapítulos anteriores, se generarán los valores de épsilon para cada una de las instancias. Se generarán los épsilon con los dos métodos propuestos para todas las instancias de Taillard con el fin de seleccionar posteriormente aquellos valores que sea más restrictivo.

Tanto en la heurística $NEH_{adaptada}$ para generar el valor de épsilon, como en la heurística $NEHFF_{adaptada}$ para generar de épsilon, se han programado en primer lugar los trabajos del conjunto A y posteriormente los trabajos del conjunto B sin que estos estén intercalados; es decir, que la secuencia final quedaría con todos los trabajos de A en las primeras posiciones y los de B en las últimas posiciones; de la siguiente manera:

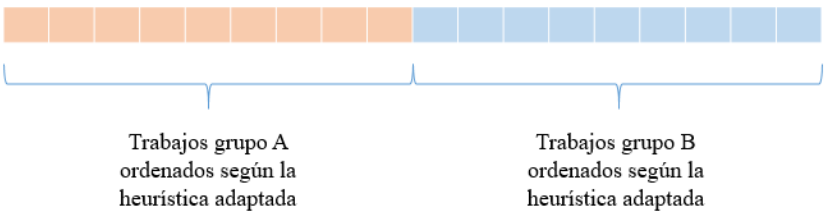

Figura 3.12: Organización de los grupos de trabajos para $NEH_{adaptada}$ y $NEHFF_{adaptada}$

En la Figura 3.12 se puede apreciar que para estas heurísticas adaptadas, el makespan de todos los trabajos será igual al makespan de B: $C_{max} = C_{maxB}$.

A la hora de seleccionar el valor de épsilon se buscará elegir un valor que pueda dar pie a encontrar soluciones factibles cuando se resuelva el problema. Es decir, un épsilon que sea lo suficientemente holgado para la restricción épsilon-constraint, ($Cmax_B \leq \varepsilon$). Posteriormente se irá reduciendo este valor de épsilon en la experimentación computacional para observar cómo se comportan las heurísticas con una restricción cada vez más limitativa. El valor de $Cmax_B$ encontrado por estas heurísticas se puede considerar holgado para esta restricción; ya que todos los trabajos de B están programados en las últimas posiciones. Basta con darse cuenta que si los trabajos de B se programarán en cualquier posición de la secuencia se podría disminuir el valor de $Cmax_B$ con facilidad. Por tanto, los épsilon se igualarán a los valores del makespan del grupo B, C_{maxB}, que obtengamos de las heurísticas adaptadas.

$$\varepsilon = Cmax_{B\text{NEHadaptada}} \qquad o \qquad \varepsilon = Cmax_{B\text{NEHFFadaptada}}$$

Se seleccionará aquella heurística que proporcione valores más restrictivos. Pero en primer lugar, los trabajos dentro de las instancias de Taillard no están clasificados en ningún conjunto, para poder aplicar estás heurísticas a las instancias es necesario conocer qué trabajos corresponden al grupo A y qué trabajos corresponden al grupo B. En este caso, se ha seleccionado que los *n/2* primeros trabajos de la instancias correspondan al conjunto A y

los *n/2* últimos trabajos correspondan al conjunto B.

Una vez seleccionados los trabajos correspondientes al grupo A y al grupo B, aplicamos las heurísticas a las instancias para poder obtener el valor del makespan de B, $Cmax_B$. Se hará un estudio de los valores que proporcionan las dos heurísticas. En la Tabla 3.3. se ha calculado la media aritmética de los valores de los makespan de B obtenidos para las dos heurísticas, para cada tamaño de instancia. Aunque las diferencias como se puede observar en la tabla no son muy grandes, se ha añadido la última columna. En la última columna se muestra cuál de las dos heurísticas consigue un valor menor, en el 75% de los casos la heurística $NEHFF_{adaptada}$ proporciona un menor valor para un mismo tipo de instancia. Es curioso observar que $NEH_{adaptada}$ ha conseguido un menor valor para instancias con el número de máquinas más elevado, m=20.

Sin embargo, para tener un marco común, solo se escogerá una heurística adaptada para generar los valores de épsilon. Se ha seleccionado la heurística $NEHFF_{adaptada}$ para generar los datos de épsilon, porque ha proporcionado menores valores medios para el makespan de B, $Cmax_B$, en la mayoría de los tipos de instancias. Por tanto, los valores de épsilon serán: $\varepsilon = Cmax_{B\text{NEHFFadaptada}}$. Estos datos se incorporarán a las instancias para poder resolver el problema de este proyecto.

Instancias	**$NEH_{adaptada}$**	**$NEHFF_{adaptada}$**	**Menor valor**
20 x 5	1353,7	1350,9	NEHFF
20 x 10	1687,9	1677,2	NEHFF
20 x 20	2484,2	2484,5	NEH
50 x 5	2890,0	2877,7	NEHFF
50 x 10	3306,5	3304,1	NEHFF
50 x 20	4184,8	4187,8	NEH
100 x 5	5437,2	5429,9	NEHFF
100 x 10	6025,8	6020,1	NEHFF
100 x 20	6986,8	6952,5	NEHFF
200 x 10	11109,6	11083,4	NEHFF
200 x 20	12260,6	12256,0	NEHFF
500 x 20	27623,4	27637,4	NEH

Tabla 3.3: Valores medios del makespan de B en la generación de épsilon

4 RESOLUCIÓN DEL PROBLEMA

En este capítulo se define la propuesta adoptada para la resolución del problema planteado en este proyecto. Una vez definido el parámetro épsilon, se tiene el punto de partida óptimo para comenzar a trabajar con el problema con de la restricción de épsilon (ε-constraint). Una vez asignados los valores de épsilon a cada una de las instancias de Taillard, se procede a resolver el problema para estudiar los diferentes resultados que se pueden conseguir según el valor de épsilon. Se propondrán tres heurísticas, todas adaptaciones de la heurística NEH original.

4.1 Adaptaciones de la heurística NEH

La heurística NEH (Nawaz et al., 1983) ha sido explicada con anterioridad en la sección 3.2.1., por lo que en este apartado se reflejarán las modificaciones realizadas al algoritmo NEH original. Según el artículo de Ruiz y Maroto (Ruiz & Maroto, 2005), donde se evaluaron diferentes heurísticas y metaheurísticas para resolver problemas con entorno *flowshop* y objetivo *makespan*, dentro de las heurísticas constructivas, NEH (Nawaz et al., 1983) era claramente más eficiente. Por esta razón ha sido seleccionada para aplicarla al problema del proyecto.

La primera modificación es para adaptar esta heurística a dos conjuntos de trabajos A y B, con la condición de que ambos grupos consten de igual número de trabajos en cada uno de los conjuntos, es decir, $n = n_A + n_B$, siendo $n_A = n_B$. Otra de las modificaciones a incluir es la restricción del parámetro épsilon, siendo este el límite superior impuesto para el valor del makespan de B, ($Cmax_B \leq \varepsilon$).

En segundo lugar, el algoritmo que se propone a continuación va generando secuencias parciales buscando la mejor solución. Puede darse la situación en la que no pueda seguir iterando el algoritmo debido a que la secuencia parcial no cumple la restricción planteada ($Cmax_B \leq \varepsilon$). Para este caso, se ha construido una *Función de Reparación* de la secuencia parcial, para poder seguir iterando, y que se cumpla la restricción de épsilon. Esta función se explicará a continuación. Esta función se presentará en Apartado 4.1.4.

También se contempla en este algoritmo que una secuencia parcial, por muchas reparaciones que se le realicen, nunca llega a ser factible. Este caso sería por ejemplo, uno en que todos los trabajos de B secuenciados estén ocupando las primeras posiciones y que se hayan probado todas las combinaciones posibles entre ellos en dichas posiciones, y ninguna de estas combinaciones cumpla la restricción de épsilon, ($Cmax_B \leq \varepsilon$). Por tanto el algoritmo devuelve un mensaje: "No existe factible para esta instancia".

Por último, a la heurística NEH adaptada para dos conjuntos de trabajos, con la restricción de épsilon y con la función reparación, se le puede modificar otro factor determinante: Aplicar distintas reglas de desempate. Las reglas de desempate se denotarán con la notación de NEH (NEH(a $|b\ |c$)). Estas reglas de desempate se utilizan para escoger la secuencia parcial para la siguiente iteración en caso de que exista un empate entre las secuencias parciales factibles con respecto al valor de la función objetivo, en este caso, el makespan de A, $Cmax_A$. Se escogerán tres reglas de desempate diferentes para construir tres adaptaciones de la heurística NEH modificada, para poder compararlas entre ellas.

4.1.1 NEH adaptada con mecanismo de desempate FS, NEH_{FS}

La primera adaptación de la heurística NEH, escogerá como regla de desempate la propia de NEH original,

presentada por Nawaz, Enscore Jr. y Ham, en 1983. Esta regla de desempate es denominada en la literatura como FS, (Fernández-Viagas et al., 2017). En caso de que dos secuencias factibles presenten un mismo mejor valor para el makespan de A, $Cmax_A$, la regla de desempate consiste en quedarse con aquella secuencia parcial factible que se haya generado en último lugar.

$$NEH\,(SD|FS|d)$$

En el pseudocódigo se pueden apreciar las modificaciones incorporadas y se pueden ver los distintos pasos que debe realizar el algoritmo de la forma más detallada posible. El código de programación elaborado para aplicar esta heurística se encuentra recogido en la sección final de Anexos de este documento, concretamente en el Anexo C.

INPUT: datos de la instancia modificada (número de máquinas, número de trabajos y matriz de tiempos de proceso, valor de épsilon)

OUTPUT: Π, $Cmax_A$, $Cmax_B$

Begin

$P_j = \sum_{i=1}^{m} P_{ij}$;

Π = ∅ , S= {1,…, n} verificando $P_1 \geq P_2 \geq \cdots \geq P_n$;

Π que sea la mejor permutación entre [1,2] y [2,1] con respecto al makespan;

Para k=3 hasta n (siendo esta la dimensión de la secuencia parcial):

Vector Posiciones_B=∅, Vector Valores_B=∅;

Se declara el *makespanA*, valor auxiliar que se inicia en un valor muy alto para comenzar el bucle;

Se declara factible=0;

Para r=0 **hasta** k-1: /*Se guardan las posiciones y los valores de los trabajos de B*/

If (Π[k]>(n/2)-1) **then**:

Posiciones_B[cont]=r;

Valores_B[cont]= Π[r];

cont ++;

While (bandera==0):

Para p=0 **hasta** k-1:

Insertar el trabajo k en la posición p de la secuencia Π;

Guardar en una nueva variable auxiliar el makespan del último trabajo de B, $Cmax_B$, en la secuencia parcial;

Guardar en una nueva variable auxiliar el makespan del último trabajo de A, $Cmax_A$, en la secuencia parcial;

If ($Cmax_A \leq makespanA\ and\ Cmax_B \leq \varepsilon$) **then:**

Se actualiza la secuencia parcial Π, el valor del makespan de A y el makespan de B;

$makespanA$=Cmax_A;

factible=1;

If ($factible == 0$ && $and\ p = k - 1$)**then:**

Se aplicará la **Función Reparación** para encontrar una secuencia parcial admisible;

If ($Contador_{combinaciones} > Combinaciones\ posibles$) **then:**

/*La heurística no es capaz de encontrar una solución factible*/

Actualizar la secuencia parcial Π;

bandera=1;

If ($factible == 1$ && $and\ p = k - 1$)**then:**

bandera=1;

Guardar la secuencia parcial Π**;**

Sí hay factible, devuelve: Π, $Cmax_A$, $Cmax_B$, $Cmax$;

Sí no hay factible, devuelve: "No existe secuencia factible para esta instancia";

Figura 4.1. NEH adaptada con mecanismo de desempate FS, NEHFS

4.1.2 NEH adaptada con un mecanismo de desempate propuesto, $\text{NEH}_{\text{propuesta}}$

En este caso se ha probado con un mecanismo de desempate muy sencillo, pero muy acorde con el objetivo y las restricciones de este problema. Esta regla de desempate se ha centrado en modificar el algoritmo basándose únicamente en la idea de minimizar el makespan de A, pero sin que el makespan de B crezca en exceso. Para ello se ha dictaminado que en caso de que dos secuencias parciales factibles tengan el mismo valor del makespan de A, $Cmax_A$; se escogerá aquella que tenga un menor valor para el makespan de B, $Cmax_B$. De aquí en adelante se le denominará regla de desempate propuesta.

$$NEH\,(SD|propuesto|d)$$

En el pseudocódigo se pueden apreciar las modificaciones incorporadas y se pueden ver los distintos pasos que debe realizar el algoritmo de la forma más detallada posible. El código de programación elaborado para aplicar esta heurística se encuentra recogido en la sección final de Anexos de este documento, concretamente en el Anexo D.

INPUT: datos de la instancia modificada (número de máquinas, número de trabajos y matriz de tiempos de proceso, valor de épsilon)

OUTPUT: Π, $Cmax_A$, $Cmax_B$

Begin

$P_j = \sum_{i=1}^{m} P_{ij}$;

Π = ∅ , S= {1,…, n} verificando $P_1 \geq P_2 \geq \cdots \geq P_n$;

Π que sea la mejor permutación entre [1,2] y [2,1] con respecto al makespan;

Para k=3 hasta n (siendo esta la dimensión de la secuencia parcial):

Vector Posiciones_B=∅, Vector Valores_B=∅;

Se declara *makespanA* y *makespanB* valores auxiliares que se inician en un valor relativamente alto;

Se declara factible=0;

Para r=0 **hasta** k-1: /*Se guardan las posiciones y los valores de los trabajos de B*/

If (Π[k]>(n/2)-1) **then**:

Posiciones_B[cont]=r;

Valores_B[cont]= Π[r];

cont ++;

While (bandera==0):

Para p=0 **hasta** k-1:

Insertar el trabajo k en la posición p de la secuencia Π;

Guardar en una nueva variable auxiliar el makespan del último trabajo de B, $Cmax_B$, en la secuencia parcial;

Guardar en una nueva variable auxiliar el makespan del último trabajo de A, $Cmax_A$, en la secuencia parcial;

If ($Cmax_A \leq makespanA$ *and* $Cmax_B \leq \varepsilon$) **then:**

If ($Cmax_A == makespanA$) **then:**

If ($Cmax_B <= makespanB$) **then:**

Se actualiza la secuencia parcial Π, el valor del makespan de A y el makespan de B;

makespanA=Cmax_A;

makespanB=Cmax_B;

factible=1;

Else:

Se actualiza la secuencia parcial Π, el valor del makespan de A y el makespan de B;

makespanA=Cmax_A;

makespanB=Cmax_B;

factible=1;

If ($factible == 0$ && $and\ p = k - 1$)**then:**

Se aplicará la **Función Reparación** para encontrar una secuencia parcial admisible;

If ($Contador_{combinaciones} > Combinaciones\ posibles$) **then:**

/*La heurística no es capaz de encontrar una solución factible*/

Actualizar la secuencia parcial Π;

bandera=1;

If ($factible == 1$ && $and\ p = k - 1$)**then:**

bandera=1;

Guardar la secuencia parcial **Π;**

Sí hay factible, devuelve: Π, $Cmax_A$, $Cmax_B$, $Cmax$;

Sí no hay factible, devuelve: "No existe secuencia factible para esta instancia";

Figura 4.2. NEH adaptada con un mecanismo de desempate propuesto, NEHpropuesta

4.1.3 NEH adaptada con mecanismo de desempate FF, NEH_{FF}

En tercer lugar se ha aplicado la regla de desempate denominada en la literatura como FF, (Fernandez-Viagas & Framinan, 2014a). Este mecanismo este relacionado con la minimización de los tiempos de inactividad total, es decir, en el caso de que los makespan de A, $Cmax_A$, de dos secuencias parciales factibles sean igual; el criterio de desempate será elegir aquella secuencia que presente un menor tiempo de inactividad total. Como se expone en la sección 3.2.3. Adaptación algoritmo NEHFF para determinar el valor de épsilon, la definición de tiempo de inactividad que se utiliza es la siguiente: En el tiempo de inactividad se considera los retrasos frontales (tiempos antes del primer trabajo) y el tiempo inactivo entre trabajos en cada máquina, pero se excluirán los retrasos posteriores (tiempo después del último trabajo en la máquina). Según esta definición se puede expresar el tiempo de inactividad de la máquina i con la siguiente ecuación, $it_i = C_{in} - \sum_{j=1}^{n} p_{ij}$ y el tiempo de inactividad total de la secuencia como $it = \sum_{i=1}^{m} it_i$.

$$NEH\,(SD|FF|d)$$

En el siguiente pseudocódigo se pueden apreciar las modificaciones incorporadas y se pueden ver los distintos pasos que debe realizar el algoritmo de la forma más detallada posible. El código de programación elaborado para aplicar esta heurística se encuentra recogido en la sección final de Anexos de este documento, concretamente en el Anexo E.

INPUT: datos de la instancia modificada (número de máquinas, número de trabajos y matriz de tiempos de proceso, valor de épsilon)

OUTPUT: Π, $Cmax_A$, $Cmax_B$

Begin

$P_j = \sum_{i=1}^{m} P_{ij}$;

Π = ∅ , S= {1,…, n} verificando $P_1 \geq P_2 \geq \cdots \geq P_n$;

Π que sea la mejor permutación entre [1,2] y [2,1] con respecto al makespan. En caso de empate se seleccionará aquella con menor tiempo de inactividad, calculando: $it_i = C_{i2} - \sum_{j=1}^{2} p_{ij}$ y posteriormente $it = \sum_{i=1}^{m} it_i$;

Para k=3 hasta n (siendo esta la dimensión de la secuencia parcial):

Vector Posiciones_B=∅, Vector Valores_B=∅;

Se declara el *makespanA*, valor auxiliar que se inicia en un valor muy alto para comenzar el bucle;

Se declara factible=0;

Determinar los valores de e_{ij}, f_{ij} *y* q_{ij}, de la aceleración de taillard para la secuencia Π;

Para r=0 **hasta** k-1: /*Se guardan las posiciones y los valores de los trabajos de B*/

If (Π[k]>(n/2)-1) **then**:

Posiciones_B[cont]=r;

Valores_B[cont]= Π[r];

cont ++;

While (bandera==0):

Para p=0 **hasta** k-1:

Insertar el trabajo k en la posición p de la secuencia Π;

Guardar en una nueva variable auxiliar el makespan del último trabajo de B, $Cmax_B$, en la secuencia parcial;

Guardar en una nueva variable auxiliar el makespan del último trabajo de A, $Cmax_A$, en la secuencia parcial;

Insertar el trabajo k en todas las posiciones desde 0…k-1 en la secuencia Π y evaluar el makespan de A;

bp= primera posición donde el makespan es mínimo de A;

tb= número de posiciones donde el makespan es mínimo;

ptb= vector donde se guardan las posiciones donde el makespan es mínimo;

$itbp$= tiempo de inactividad correspondiente a bp, se fija en un valor relativamente grande;

If (tb>1 && k<n/2**) then**

Para l=1 hasta tb:

It''=0;

If (ptb[l]=k**) then:**

Para i=2 hasta m:

$$It'' = It'' - e_{i,k-1} + f_{ik} - t_{ir};$$

Else:

$$f'_{1,ptb[l]} = f_{1,ptb[l]} + p_{1,ptb[l]};$$

Para i=2 hasta m:

$$it'' = it'' + f_{i,ptb[l]} - e_{i,ptb[l]} + p_{i,ptb[l]} - t_{ir} + max\left\{f'_{i-1,ptb[l]} - f_{i,ptb[l]},\ 0\right\};$$

$$f'_{i,ptb[l]} = max\left\{f_{i,ptb[l]}, f'_{i-1,ptb[l]}\right\} + p_{i,ptb[l]};$$

If ($itbp < it''\ and\ Cmax_B \leq \varepsilon$**) then:**

bp=ptb[l];

itbp=it'';

Se actualiza la secuencia parcial Π, insertando el trabajo k en la posición bp;

Se actualiza el valor del makespan de A y el makespan de B;

factible=1;

If ($factible == 0$ && $and\ p = k - 1$)**then:**

Se aplicará la **Función Reparación** para encontrar una secuencia parcial admisible;

If ($Contador_{combinaciones} > Combinaciones\ posibles$) **then:**

/*La heurística no es capaz de encontrar una solución factible*/

Actualizar la secuencia parcial Π;

bandera=1;

If ($factible == 1$ && $and\ p = k - 1$)**then:**

bandera=1;

Guardar la secuencia parcial Π**;**

Sí hay factible, devuelve: Π, $Cmax_A$, $Cmax_B$, *Cmax;*

Sí no hay factible, devuelve: "No existe secuencia factible para esta instancia";

Figura 4.3. NEH adaptada con mecanismo de desempate FF, NEHFF

4.1.4. Función Reparación

La Función Reparación que se ha planteado con la finalidad de modificar y reparar las secuencias parciales infactibles, para poder continuar con la construcción de la secuencia en las tres versiones de la NEH presentadas anteriormente. Esta reparación solo será necesaria cuando en una iteración del algoritmo no se encuentre ninguna secuencia parcial factible al introducir el trabajo k en cada una de las posiciones posibles (p=0 hasta p=k-1). Al no encontrarse ninguna solución factible se debe modificar la secuencia parcial anterior, es decir, sin introducir este nuevo trabajo k. Por tanto, se modifica la secuencia parcial con la dimensión k-1.

Que una secuencia parcial sea factible o no, depende de si cumple la restricción épsilon (ε-*constraint*). En este caso que el makespan de B sea menor a un valor fijado épsilon, ε, ($Cmax_B \leq \varepsilon$). La mecánica de esta Función Reparación se basa en ir desplazando los trabajos de B hasta posiciones anteriores en la secuencia; es decir, posiciones cada vez más cercanas al inicio, para así obtener menor makespan de B, $Cmax_B$. Los pasos a seguir por esta función son los siguientes:

1. En primer lugar se pasa la secuencia parcial de dimensión k-1 a la función y un vector con las posiciones de los trabajos de B en la secuencia parcial $\Pi_{posicionesB}$. Este vector se utiliza para saber la posición máxima en la que se puede introducir cada trabajo de B.
2. La función identifica el último trabajo de B fijándose en el vector de posiciones de los trabajos de B, $\Pi_{posicionesB}$. Una vez identificado, se extrae de la secuencia parcial de dimensión k-1.
3. Se realiza un bucle en el que se va introduciendo dicho trabajo en todas las posiciones posibles. Estas posiciones posibles son desde p=0 hasta la posición indicada en el vector posiciones de B, $\Pi_{posicionesB,}$ (sin incluir este último). Es decir, posiciones anteriores a la que se encontraba. Se elegirá la posición en la que el makespan de A, $Cmax_A$, sea menor dentro de los factibles.
4. Una vez que se construye la nueva secuencia parcial, al haber cambiado el último trabajo de B, se actualiza el valor del vector de posiciones de B, $\Pi_{posicionesB.}$ Es decir, se mete en el vector, $\Pi_{posicionesB}$, la nueva posición de dicho trabajo de B. Esta actualización se hace para que no se produzcan ciclos, porque así cada vez que se repita este mismo trabajo de B se irá viendo obligado a ocupar posiciones más a la izquierda en la secuencia parcial.
5. Como se ha indicado en el algoritmo anterior también se contempla que una secuencia parcial por muchas reparaciones que se le realicen nunca llegue a ser factible. Este caso sería aquel en que todos los trabajos de B secuenciados estén en las posiciones más a la izquierda y que se hayan probado una serie de combinaciones entre ellos en dichas posiciones, y ninguna de estas combinaciones cumpla la restricción de épsilon, ($Cmax_B \leq \varepsilon$). Esto se realiza dentro de la función reparación con un contador. Que lo que hace es contar las combinaciones que se han probado con todos los trabajos de B a la izquierda. El contador se fijará en función del número de trabajos, el contador se fijará en el valor de *n*. Lo correcto sería introducir como máximo todas las permutaciones posibles entre los n_B trabajos de B, es decir, el factorial de *n*, *n!*. No se ha tomado este valor porque los valores que se obtendrían serían demasiado grandes para realizarlos en un tiempo razonable en la experimentación computacional. Por ejemplo, para n_B=10, el número de permutaciones asciende a un valor de 7 cifras.

En el siguiente pseudocódigo se expresan los distintos pasos que debe realizar el algoritmo de la función. El código de programación elaborado para la aplicación de esta función se encuentra recogido en la sección final de Anexos de este documento, concretamente en el Anexo F. La terminología utilizada es la misma que en los pseudocódigos anteriores, ya que esta función es realmente una parte de los algoritmos anteriores.

INPUT:

Número de máquinas, número de trabajos y matriz de tiempos de proceso así como el valor de épsilon;

Vector LPT (Longest Processing Time) de todos los trabajos;

Secuencia parcial generada y su dimensión;

Trabajos de B en la secuencia y sus posiciones en la misma;

Contador de veces que se entra en la función reparación para una misma dimensión de la secuencia parcial;

OUTPUT:

Secuencia reparada;

Begin

Se selecciona el último trabajo de B que aparece en la secuencia parcial que entra en la función;

Se guarda en una variable auxiliar (aux) la posición más a la derecha en la que este trabajo de B puede introducirse, que es la correspondiente a ese trabajo en el vector Posiciones_B; (En la primera iteración será la posición en la secuencia parcial original)

Para p=0 **hasta** k-1:

Insertar el trabajo de B en la posición p de la secuencia parcial Π;

Guardar en una nueva variable auxiliar el makespan del último trabajo de B, $Cmax_B$, en la secuencia parcial;

Guardar en una nueva variable auxiliar el makespan del último trabajo de A, $Cmax_A$, en la secuencia parcial;

If ($Cmax_A \leq makespanA\ and\ Cmax_B \leq \varepsilon$) **then:**

Actualizamos la secuencia parcial Π, el valor del makespan de A y el valor del makespan de B;

factible=1;

If ($factible == 0$ && $and\ p = aux\ and\ aux = 0$)**then:**

$Contador_{combinaciones}$++;

Insertar el trabajo en la posición aux;

If ($factible == 0$ && $and\ p = aux\ and\ aux! = 0$)**then:**

/*Guarda la posición actual del trabajo de B en la secuencia modificada, en el vector Posiciones_B*/

Posiciones_B [trabajo B]=Posición del último trabajo de B en $\Pi_{modificada}$;

Copiar el vector $\Pi_{modificada}$ en el vector Secuencia Reparada;

Devuelve: Secuencia Reparada y actualiza el $Contador_{combinaciones}$ y el vector Posiciones_B;

Figura 4.4. Código Función Reparación

4.2 Experimentación computacional

En este capítulo se resolverán todas las Instancias de Taillard con las diferentes heurísticas presentadas para analizar posteriormente los datos obtenidos. Se presentarán los distintos parámetros con los que se pueden resolver las instancias e incluso se irá forzando la restricción de *épsilon-constraint* para hacerla cada vez más dura. Con todas estas modificaciones que se pueden incluir, se estudiará la progresión de los algoritmos. Dado que la función objetivo que se trata de minimizar es el makespan del grupo A, $Cmax_A$, se guardará dicho valor para cada instancia. Este valor del makespan de A corresponderá a la mejor solución factible encontrada por el algoritmo. Sin embargo, si los algoritmos no encuentran ninguna secuencia factible; devolverá un mensaje comentando esta casuística y dando por terminada la búsqueda.

Para poder resolver el problema de este proyecto se hará uso de las heurísticas y códigos presentados en el capítulo anterior. Aplicados a las Instancias de Taillard modificadas, (Taillard, 1993). Los valores de épsilon han sido explicados en la sección 3.2.4. Los valores de épsilon son los generados por la heurística $NEHFF_{adaptada}$ para los dos grupos de trabajos (subcapítulo 3.2.3.). De ahora en adelante se denominarán a estos valores como: $\varepsilon_{CmaxB_{NEHFF}}$.

Para poder ver el impacto del valor de épsilon en las heurísticas su valor se irá reduciendo. Para reducir el valor de épsilon se hará uso de una variable auxiliar que se denotará como α. Por tanto, épsilon, ε, para la restricción *épsilon-constraint, ($Cmax_B \leq \varepsilon$)*, será calculado a través de la siguiente ecuación:

$$\varepsilon = (1 - \alpha) * \varepsilon_{CmaxB_{NEHFF}}$$
$$\alpha \in [0,1]$$

A medida que α vaya creciendo, el valor de épsilon irá disminuyendo lo que significa que la restricción *épsilon-constraint* será más restrictiva con cada valor de α. Lo que dará lugar cada vez a un número menor de sencuencias factibles. Los valores de alfa que se van a utilizar son: $\alpha = 0; \alpha = 0{,}05;\ \alpha = 0{,}10;\ \alpha = 0{,}20;\ \alpha = 0{,}30;\ \alpha = 0{,}40;\ \alpha = 0{,}50$. Las baterías creadas con las instancias se resolverán para estos siete valores de α definidos. Por tanto, el valor más holgado de la restricción se conseguirá con $\alpha = 0$, lo que es lo mismo que decir $\varepsilon_{\alpha=0} = \varepsilon_{CmaxB_{NEHFF}}$ y el valor más restricctivo con $\alpha = 0{,}5$, donde épsilon será el 50% del valor original, por tanto, $\varepsilon_{\alpha=0{,}50} = 0.5 * \varepsilon_{CmaxB_{NEHFF}}$. Es fácil ver que con $\alpha = 0$, todas las instancias serán factibles dada la manera en la que se ha obtenido $\varepsilon_{CmaxB_{NEHFF}}$. Esto se debe a que como el problema busca minimizar el makespan de A, esté conjunto no estará muy limitado si el valor de épsilon es muy holgado. La idea de probar diferentes valores de épsilon, más restrictivos (al disminuir el valor de épsilon, el makespan de B disminuye) es ver como se ve afectado el makespan de A. Será interesante ver el porcentaje de empeoramiento del makespan de A a medida que se va mejorando el de B.

En primer lugar se tiene que tener claro de qué tres factores dependerán los resultados obtenidos:

- Mecanismos de desempate: (3) Mecanismo de desempate de FS, Mecanismo de desempate propuesto y Mecanismo de desempate FF.
- Tipos de instancias: (12) Existen doce tipos diferentes de instancias con respecto al número de trabajos y al número de máquinas (Tabla1: Distribución de máquinas y trabajos en las Instancias de Taillard).
- α: (7) valores de α con los que se modificará la restricción de épsilon. *{0; 0,05; 0,1; 0,2; 0,3; 0,4; 0,5}*.

Sin embargo, una vez se realizó la experimentación computacional para $\alpha = 0{,}50$ el porcentaje de infactibilidad fue del 100%. Es decir, no se encontró ninguna solución factible para este valor α con ninguna de las tres heurísticas propuestas en este proyecto.

Para poder comparar los tres métodos aplicados es necesario se van a calcular los ARPD de cada algoritmo que se ha probado en este proyecto. $ARPD_j$ es el valor de la desviación porcentual relativa promedio para cada

algoritmo j, (Average Relative Percent- age Deviation). Se calcula con la siguiente fórmula:

$$\text{ARPD}_j = \frac{\sum_{i=0}^{I} RPD_{i,j}}{I}$$

Donde el valor de I es el número de instancias para las que se han podido obtener los valores de la desviación porcentual relativa, RPD, (Relative Percentage Deviation). Siendo $RPD_{i,j}$ la desviación porcentual relativa obtenida por el algoritmo j cuando se aplica a la instancia i y generalmente se calcula de la siguiente manera:

$$RPD_{i,j} = \frac{C_{max,i,j} - Best_i}{Best_i} * 100$$

Donde $C_{max,i,j}$ es el makespan que se ha obtenido con el algoritmo j en la instancia i y $Best_i$ es el límite superior (la mejor solución conocida) para esa instancia. Sin embargo, como el objetivo de este problema era minimizar el makespan de A se tomará $C_{max,i,j}$ como $C_{maxA,i,j}$, es decir, el makespan de A en vez del makespan de todos los trabajos de la instancia.

En el artículo de Víctor Fernández-Viagas, Rubén Ruiz y José M. Framiñan, (Fernandez-Viagas et al., 2017), para seleccionar el valor de $Best_i$ han escogido los límites superiores fijados por Taillard (Taillard, 1993) para las instancias correspondientes. Pero estos límites han sido hallados con respecto al makespan de todos los trabajos, sin embargo, en este proyecto el objetivo a minimizar es el makespan únicamente del subconjunto A con una restricción sobre el makespan de grupo B. Para poder fijar unos datos de referencia que se puedan utilizar en este proyecto se tomará como valor de $Best_i$, el menor resultado encontrado de entre las tres reglas de desempate para cada instancia. Es decir, $Best_i$ será el menor makespan de A que se haya encontrado para la instancia i.

Para poder analizar y obtener datos de este problema se han programado una serie de algoritmos que se han ido explicando a lo largo de este documento, están recogidos en la sección de Anexos de este documento. Para la programación de estos algoritmos y la ejecución de los mismos se ha empleado Code::Blocks, un software libre. El lenguaje de programación utilizado en este caso es C++. (*Code :: Blocks*, n.d.)

Este programa nos permite guardar los códigos como archivos ejecutables para utilizarlos en el Símbolo del sistema de Windows (Command Prompt), y poder recoger los resultados de las instancias que queremos ejecutar. Una vez ejecutados los códigos de Code::Blocks se extraerán los resultados obtenidos de las variables necesarias para hacer un análisis de datos. En este caso aunque se podrían extraer todas las variables utilizadas, solo se guardarán aquellas que son de mayor interés. Por tanto, se guardará el valor de la función objetivo que es el valor del makespan de A, $Cmax_A$; también se guardará el valor del makespan de B, con el fin de discernir si se cumple o no la restricción de épsilon *(ε-constraint*). Por otra parte, se guardará una variable auxiliar que reportará si el algoritmo ha encontrado o no una secuencia que se pueda considerar solución factible al problema.

También se ha utilizado Microsoft Excel para guardar y analizar los resultados obtenidos. (*Microsoft Excel, Software de Hojas de Cálculo*, n.d.)

4.2.1 Comparación de los algoritmos con respecto de α

Los resultados obtenidos de todas las instancias para cada uno de los valores del parámetro α se recogen en la Tabla 4.1. que se muestra a continuación. No se han incluido en esta tabla los resultados para $\alpha = 0{,}50$ ya que ningún método proporciona ninguna solución factible para ninguna instancia. En la siguiente tabla el valor que aparece en cada celda es la media aritmética obtenida de las desviaciones porcentuales relativas ($ARPD_{i,j}$) de las 10 instancias pertenecientes a cada grupo. También se tiene que remarcar que para α *={0,30; 0,40}* no se ha conseguido encontrar una solución factible (Feasible Schedule) para todas las instancias con las iteraciones que había prefijadas. Por tanto, la media aritmética de las desviaciones porcentuales relativas ($ARPD_{i,j}$) no se ha podido hacer en todos los tamaños con 10 instancias, sino con un número menor. En el eje vertical en el que se agrupan los diferentes tipos de instancias, el primer parámetro hace referencia al número de trabajos de la instancia y el segundo al número de máquinas de la misma, *n* x *m*.

		α																	
n	m	0			0,05			0,10			0,20			0,30			0,40		
		NEH_{FS}	NEH_{prop}	NEH_{FF}	NEH_{FS}	NEH_{prop}	NEH_{FF}	NEH_{FS}	NEH_{prop}	NEH_{FF}	NEH_{FS}	NEH_{prop}	NEH_{FF}	NEH_{FS}	NEH_{prop}	NEH_{FF}	NEH_{FS}	NEH_{prop}	NEH_{FF}
20	**5**	47,618	5,361	32,385	12,009	9,169	6,336	2,976	0,469	2,302	1,005	1,289	1,466	2,161	1,058	1,803	2,572	2,303	1,260
	10	30,561	0,865	23,482	8,062	1,506	7,021	4,822	0,574	3,286	1,642	1,307	1,740	2,865	2,043	0,319	0,000	0,000	1,380
	20	15,283	0,738	22,662	1,506	0,866	3,172	1,363	1,117	1,488	0,912	1,807	2,371	0,000	0,630	0,295	-	-	-
50	**5**	67,306	0,000	76,305	0,994	0,264	2,593	1,680	0,668	1,640	1,630	0,796	1,522	2,621	0,406	1,584	0,870	2,739	0,672
	10	46,943	2,008	47,331	1,489	1,958	1,509	0,897	0,607	1,216	2,297	0,526	1,028	1,419	2,177	0,954	0,763	1,119	0,699
	20	38,374	0,025	38,484	1,629	0,570	1,724	1,874	0,413	1,364	1,251	1,125	1,364	1,391	0,996	2,904	-	-	-
100	**5**	57,584	0,000	58,500	1,720	0,339	1,326	0,790	0,620	0,897	0,763	0,497	0,297	0,578	0,773	0,777	1,543	1,310	1,591
	10	73,049	3,614	68,390	2,082	0,844	1,743	1,695	0,662	1,984	1,074	0,752	0,662	1,673	0,743	0,849	1,369	1,518	0,590
	20	39,580	0,489	39,695	1,038	0,440	0,566	1,094	0,672	0,947	0,778	0,263	1,316	0,860	1,294	0,508	3,437	0,000	3,598
200	**10**	57,414	0,143	57,795	0,970	0,261	1,043	0,851	0,452	1,004	1,017	0,571	0,512	0,734	0,077	0,280	1,946	0,584	1,133
	20	36,294	0,116	36,524	0,855	0,548	2,307	1,097	0,414	0,656	0,412	0,735	0,591	0,539	0,752	0,526	0,536	0,711	0,373
500	**20**	29,241	0,017	30,382	29,295	0,000	30,130	0,072	0,000	0,972	-	-	-	-	-	-	-	-	-

Tabla 4.1: Desviación porcentual relativa promedio de las diferentes heurísticas en función del valor de α y del tamaño de instancia

Como se puede observar en la Tabla 4.1. para $\alpha=0,40$, existen dos filas sin datos asignados; las correspondientes a los grupos de instancias *20x20* y *50x20.* Esto se debe a que ninguno de los tres algoritmos no ha sido capaz de encontrar una solución factible para ninguna de las 10 instancias correspondientes a los tamaños *20x20* y *50x20*. Para hacer un análisis de los resultados obtenidos es importante remarcar que cuando se tiene una menor desviación porcentual relativa promedio significa mejores resultados para dicho mecanismo de desempate. Otro dato a destacar de la Tabla 4.1 es que en los caso con $\alpha = \{0; 0,05; 0,10; 0,30; 0,40\}$, es que tienen celdas cuyo valor es 0,000; esto se debe a que ese mecanismo de desempate ha conseguido el mejor resultado del makespan de A, $Cmax_A$ para todas las instancias de un mismo tamaño.

Observando la Tabla 4.1. con $\alpha = 0;$ llama la atención por la disparidad de los valores de ARPDj y la gran diferencia existente entre los valores encontrados para NEH_{FS} y NEH_{FF} comparación con la heurística que se ha denominado como $NEH_{PROPUESTA}$. Estas grandes diferencias se pueden apreciar con facilidad en la Figura 4.5., esto significa que el mecanismo de desempate propuesto funciona mucho mejor que los otros dos con $\alpha = 0$ para todos los tamaños de instancias de Taillard que se están utilizando para el problema propuesto.

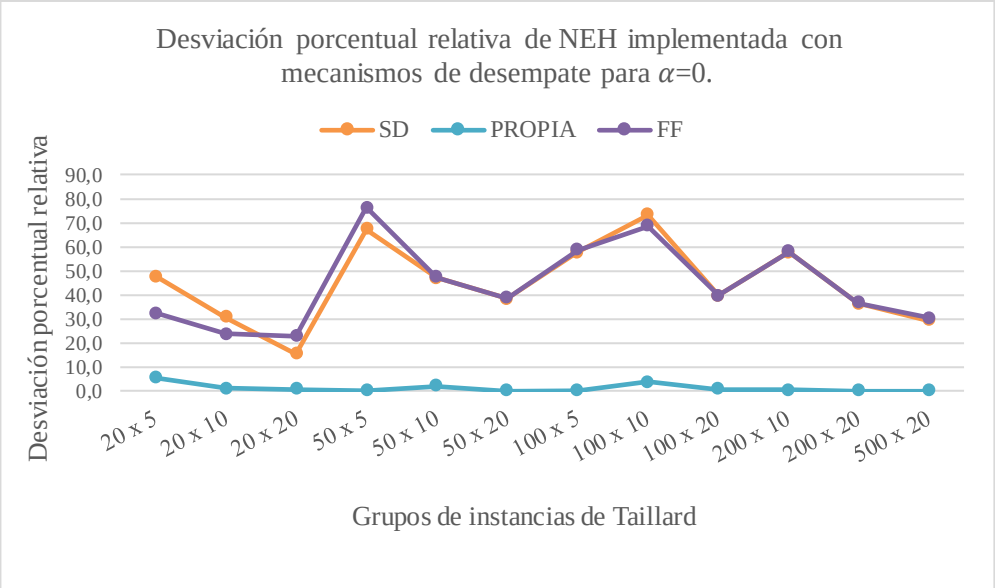

Figura 4.5: Desviación porcentual relativa de NEH implementada con mecanismos de desempate para α =0

Ahora se dejarán aparte los valores para $\alpha = 0$. Se dejarán aparte porque ya se han analizado en la Figura 4.5 anterior y porque tienen un rango numérico muy superior respecto a los valores conseguidos de ARPDj para los demás valores de α y por tanto dificultan la visualización gráfica de los demás datos. En la Figura 4.6 se puede apreciar las diferencias entre los datos obtenidos por las tres heurísticas para cada uno de los valores del parámetro α que se están analizando. La primera observación es que el mecanismo de desempate Propuesto obtienen un menor valor de la desviación promedio en 4 de los 5 casos. Si se cuenta que en $\alpha = 0$ también es el mejor, se pondría en 5 de 6 casos, es decir, es el mejor para el *83,30 %* de los valores de α.

Por otra parte, se observa que existe un mayor abanico en el rango de valores para las α más pequeños. A medida que se va restringiendo el valor de épsilon (es decir, a medida que va creciendo el valor de α) también se restringe la variación en los valores de la desviación porcentual relativa promedio, ARPDj; es decir, hay menos diferencias entre los resultados obtenidos por los tres mecanismos de desempate. Si se analiza el problema que se intenta resolver en este proyecto es lógico que se dé esta situación. A medida que la restricción de *épsilon-constrait* va haciéndose cada vez más restrictiva, la maniobrabilidad de los algoritmos se va reduciendo y por tanto, los resultados van tendiendo a un rango de valores más reducido. Otro aspecto a destacar es que el mecanismo de desempate FF funciona mejor para el caso más restrictivo que se estudia en este proyecto, para $\alpha = 0{,}40$.

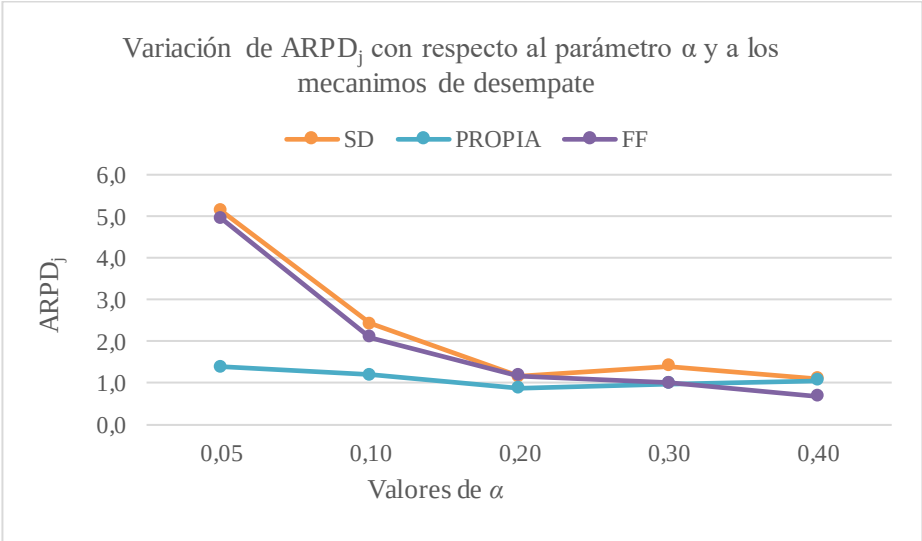

Figura 4.6: Variación de ARPDj con respecto al parámetro α y a los mecanimos de desempate

Otra parte interesante a analizar de este problema es la capacidad de los diferentes algoritmos para encontrar secuencias factibles (Feasible Schedule) dentro de un número de iteraciones prefijadas con antelación. Los porcentajes de soluciones factibles encontradas aparecen en la Tabla 4.2 según el valor de α y la heurística utilizada. Junto a esta información también aparece la desviación porcentual relativa promedio total para cada heurística propuesta contando con todas las instancias que se han evaluado. No se ha incluido el valor de *α=0,50* ya que no se pueden proporcionar los datos para ARPDj totales.

	α											
	0		**0,05**		**0,10**		**0,20**		**0,30**		**0,40**	
	ARPD	%fact	ARDP	%fact	ARDP	%fact	ARDP	%fact	ARDP	%fact	ARDP	%fact
NEH_{FS}	44,937	100%	5,137	100%	1,601	100%	1,069	100%	1,299	93%	1,051	64%
NEH_{prop}	1,115	100%	1,397	100%	0,601	93%	0,871	93%	1,042	86%	1,051	63%
NEH_{FF}	44,328	100%	4,956	100%	1,511	94%	1,170	92%	0,997	86%	0,706	58%

Tabla 4.2: Valores de ARPDj totales y porcentaje de factibilidad de las heurísticas en función de α

En la Figura 4.7. se observa como va disminuyendo el porcentaje de factiblidad a medida que va creciendo el valor de α, que es lo mismo que decir, a medida que va disminuyendo el valor de épsilon, ε.

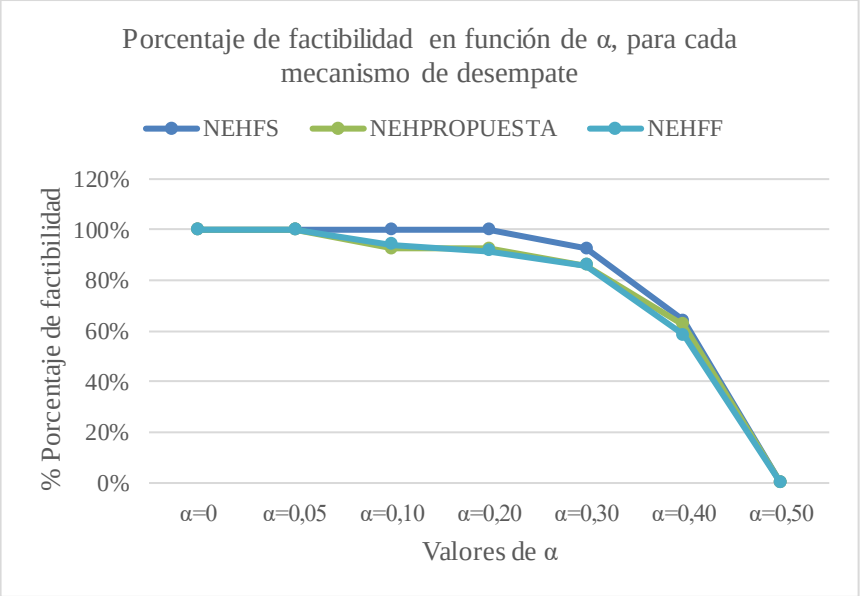

Figura 4.7: Porcentaje de factibilidad en función de α, para cada mecanismo de desempate

4.2.2 Comparación de los algoritmos con respecto de m y n

Las instancias probadas en este proyecto tienen variabilidad con respecto al número de trabajos y de máquinas, por tanto, otro punto de análisis en este proyecto puede ser centrándose en las diferencias que generan estas características de las instancias. . Se presentará las gráficas que describen el valor de la desviación porcentual relativa promedio, $ARPD_j$, para cada valor de α, en función del número de máquinas y en función del número de trabajos para cada una de las heurísticas adaptadas.

En primer lugar se hará un análisis en función del número de máquinas, m. En la Figura 4.8 se presentará como varía el valor de $ARPD_j$ en función del número de máquinas para NEH_{FS}; en la Figura 4.9 y Figura 4.10 se plasmará la misma información pero para $NEH_{PROPUESTA}$ y NEH_{FF} respectivamente.

Se observa claramente que el comportamiento de NEH_{FS} y NEH_{FF} es muy parecido, y sin embargo, para $NEH_{PROPUESTA}$ el comportamiento en función del número de máquinas es muy diferente, basta con fijarse en los valores de los ejes de las gráficas. Llama la atención que el valor de la desviación porcentual relativa promedio en función del número de máquinas, es muy superior para $\alpha = 0$, menos para $NEH_{PROPUESTA}$. Para $NEH_{PROPUESTA}$ para $\alpha = 0$ se observa una caída abrupta con m=20. Para los demás valores de α la desviación promedio aparte de tener valores mucho más bajos también varía mucho menos en función del número de máquinas.

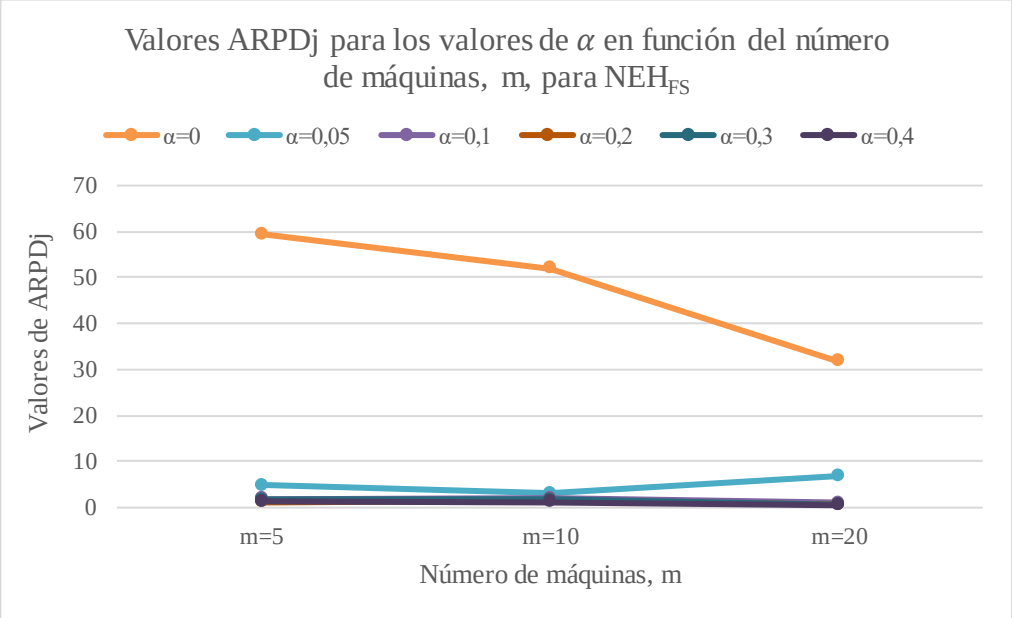

Figura 4.8: Valores ARPDj para los valores de α en función del número de máquinas, m, para NEH_{FS}

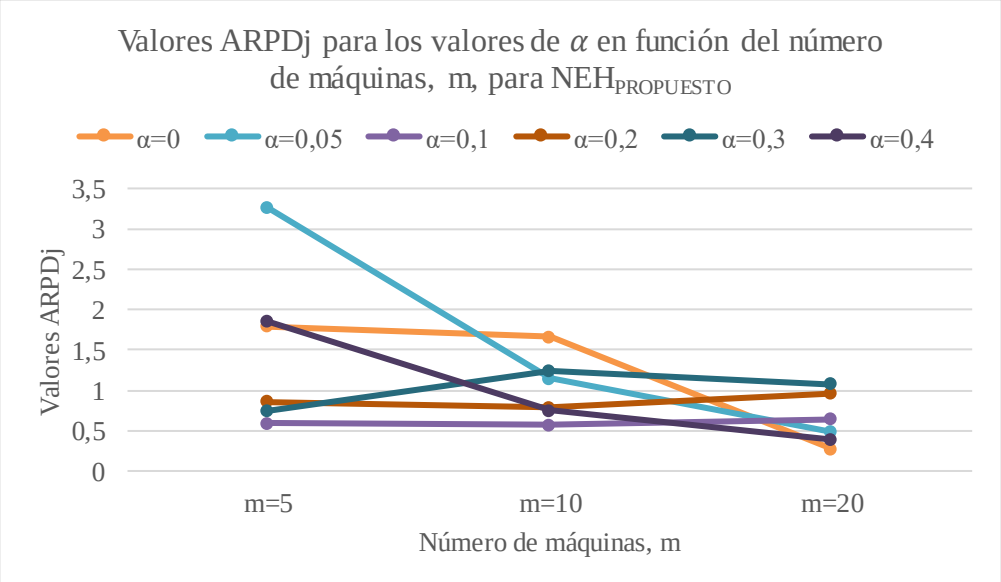

Figura 4.9: Valores ARPDj para los valores de α en función del número de máquinas, m, para $NEH_{PROPUESTO}$

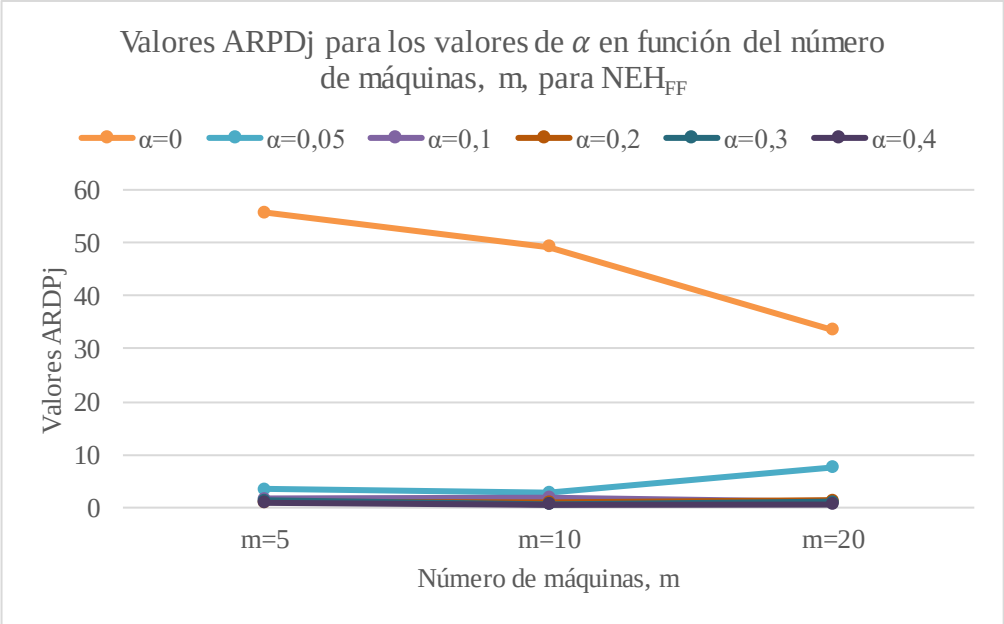

Figura 4.10: Valores ARPDj para los valores de α en función del número de máquinas, m, para NEH_{FF}

Tanto para NEH_{FS} como para NEH_{FF}, la progresión de los resultados con los diferentes valores de α se puede apreciar con menos exactitud en estas gráficas debido a la gran diferencia de valores, sin embargo la tendencia es decreciente, aunque no tanto como para $NEH_{PROPUESTA}$. La tendencia decreciente para esta heurística se puede ver claramente en la Figura 4.9. Esto significa que en general con independencia del valor de α, cuanto más grande es el número de máquinas, menores valores de ARPDj se obtienen. Por tanto, los tres algoritmos funcionan mejor cuanto mayor número de máquinas hay en el sistema.

En segundo lugar se hará un análisis en función del número de máquinas, m. En la Figura 4.11 se presentará como varía el valor de $ARPD_j$ en función del número de trabajos para NEH_{FS}; en la Figura 4.12 y Figura 4.13 se plasmará la misma información pero para $NEH_{PROPUESTA}$ y NEH_{FF} respectivamente. De nuevo lo primero que se puede comprobar es que para la regla de desempate FS y la regla de desempate FF, el comportamiento es muy similar.

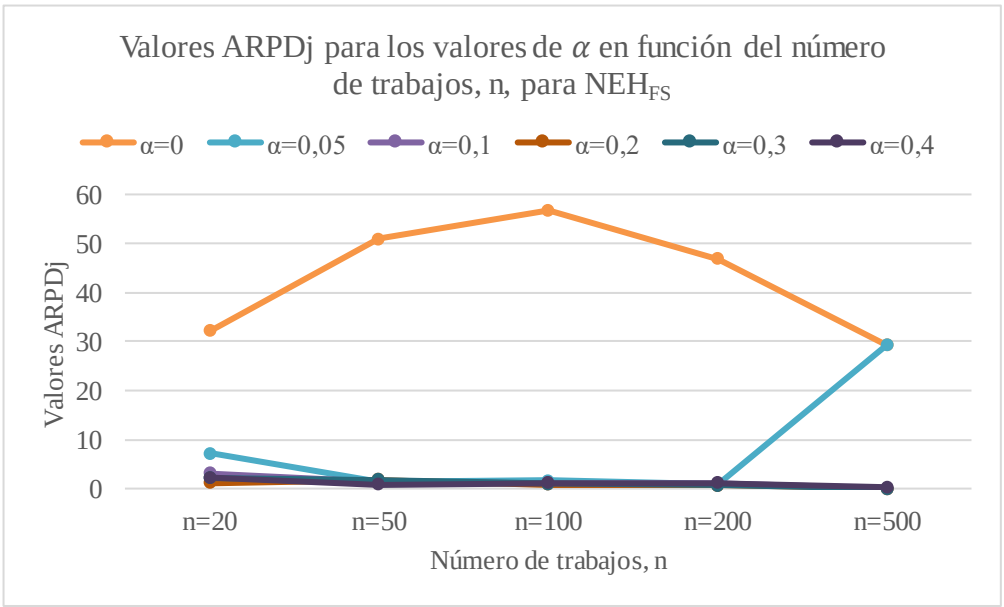

Figura 4.11: Valores ARPDj para los valores de α en función del número de trabajos, n, para NEH_{FS}

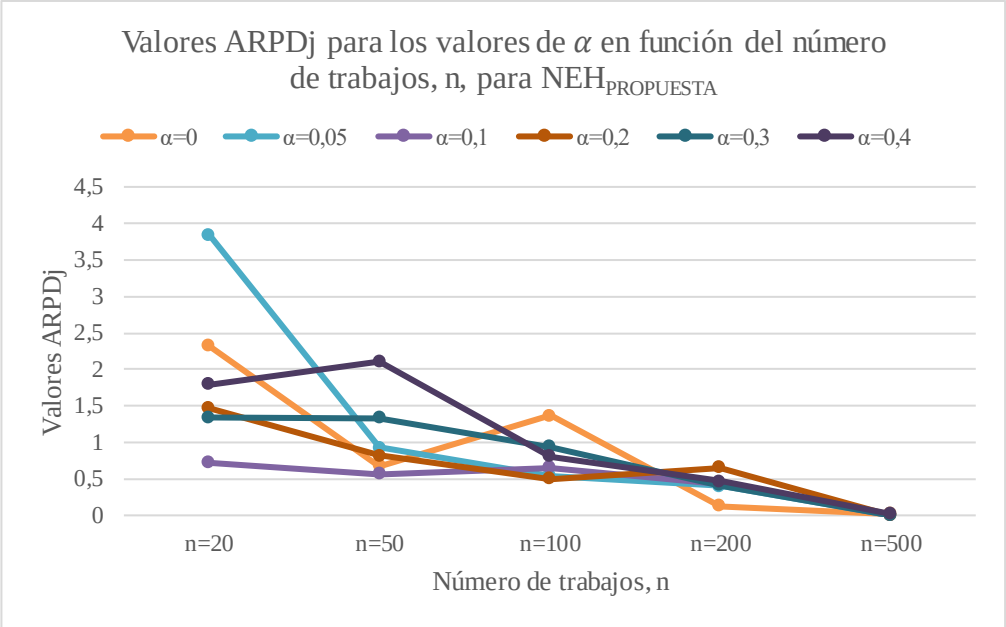

Figura 4.12: Valores ARPDj para los valores de α en función del número de trabajos, n, para $NEH_{PROPUESTA}$

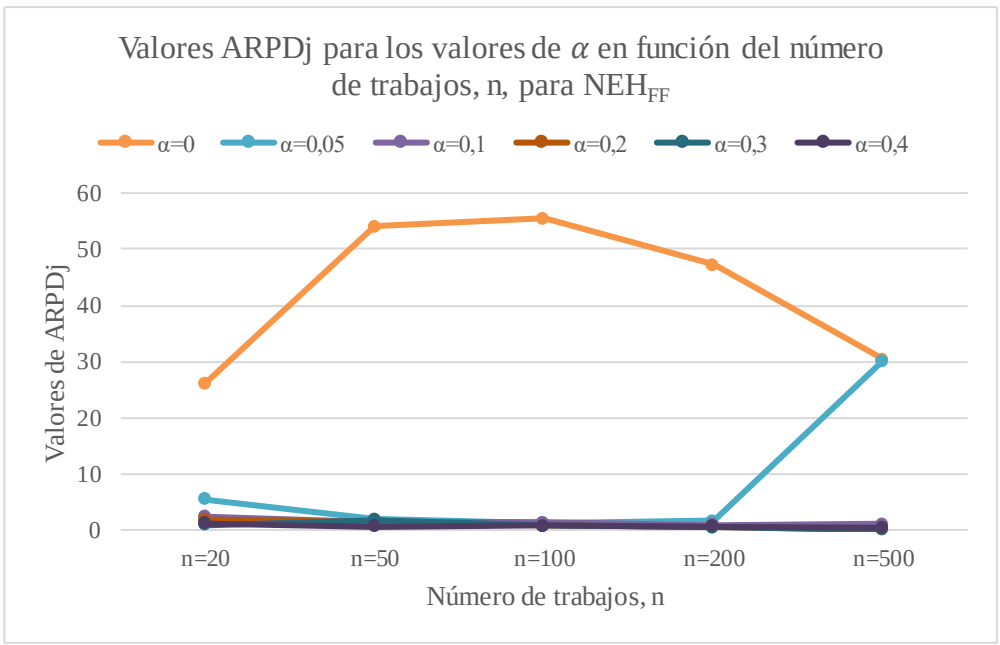

Figura 4.13: Valores ARPDj para los valores de α en función del número de trabajos, n, para NEH_{FF}

Tanto para la Figura 4.11como para la Figura 4.13. se aprecia que la curva para $\alpha = 0$ no tiene una tendencia clara; ya que va creciendo hasta n=100 y a partir de ahí decrece para instancias con mayor número de máquinas. Para $\alpha = 0{,}05$ se aprecia para estas dos misma gráficas que tiene un comportamiento decreciente, salvo en n=500, donde crece muy rápidamente. Quitando estas dos excepciones lo que se puede apreciar es que para valores de m mayores, los resultados van siendo mejores. Esto se ve más claramente en la Figura 4.12 para $NEH_{PROPUESTA}$, donde con independencia de α, la tendencia es decreciente a medida que aumenta el número de trabajos.

Los peores resultados en comparación con aquel que se ha fijado como $Best_i$ se dan para los valores de α más pequeños $\alpha = 0$ y $\alpha = 0{,}05$, sobre todo para las heurísticas NEH_{FS} y NEH_{FF}. Sin embargo hay que tener en cuenta que a medida que crece el valor de α y la restricción del problema se va haciendo más limitativa, las desviaciones promedio tienen menor rango de desviación y van disminuyendo.

		α																	
		0			0,05			0,10			0,20			0,30			0,40		
n	m	NEH_{FS}	NEH_{prop}	NEH_{FF}	NEH_{FS}	NEH_{prop}	NEH_{FF}	NEH_{FS}	NEH_{prop}	NEH_{FF}	NEH_{FS}	NEH_{prop}	NEH_{FF}	NEH_{FS}	NEH_{prop}	NEH_{FF}	NEH_{FS}	NEH_{prop}	NEH_{FF}
20	5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	30%	20%	30%
	10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	10%	10%	10%	90%	90%	90%
	20	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	70%	60%	60%	100%	100%	100%
50	5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	10%	10%
	10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	20%	30%	20%
	20	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	10%	0%	0%	100%	100%	100%
100	5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	20	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	70%	60%	60%
200	10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	10%	0%	0%
	20	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
500	20	0%	0%	0%	0%	0%	0%	0%	90%	70%	50%	100%	90%	50%	100%	100%	60%	40%	90%

Tabla 4.3. Porcentaje de infactibilidad de las diferentes heurísticas en función del valor de α y del tamaño de instancia

Por otra parte, es interesante también hacer un análisis del porcentaje de factibilidad según el número de máquinas y el número de trabajos. El porcentaje de infactibilidad según el tipo de heurística y para cada valor de α, está reflejado en la Tabla 4.3. En primer lugar, se tiene que hacer diferencia entre: No encontrar una secuencia factible dentro de las iteraciones prefijadas y que el algoritmo devuelva que aún habiendo probado todas las secuencias que puede generar no hay ninguna que sea factible. La Figura 4.14. muestra el porcentaje de instancias para las que no se ha encontrado una solución factible en función del número de máquinas para las tres heurísticas propuestas; de igual manera la Figura 4.15 muestra la información correspondiente al número de trabajos. En primer lugar se explicarán como se han realizado estas gráficas. Para comprender la información que proporcionan en primer lugar, se debe recordar que no todos los tamaños de máquinas y de trabajos tienen las mismas instancias asignadas. Por tanto, esto se ha tenido en cuenta a la hora de realizar el porcentaje de infactibilidad. También en estas gráficas se agruparán los datos para todos los valores de α.

Se observa que la tendencia es creciente con respecto al número de máquinas, para cualquier número de trabajos. Los algoritmos tienen más dificultades para encontrar una solución factible en las instancias que cuentan con un mayor número de máquinas, *m*=20. Observando la Tabla 4.3. y la Figura 4.14. se confirma esta tendencia, ya que la mayoría de los porcentajes de infactibilidad altos se dan para *m*=20. Sin embargo, se observa una anomalía para *m*=20 con *n*=200, para este caso todos las heurísticas son capaces de encontrar una solución factible para todas las instancias.

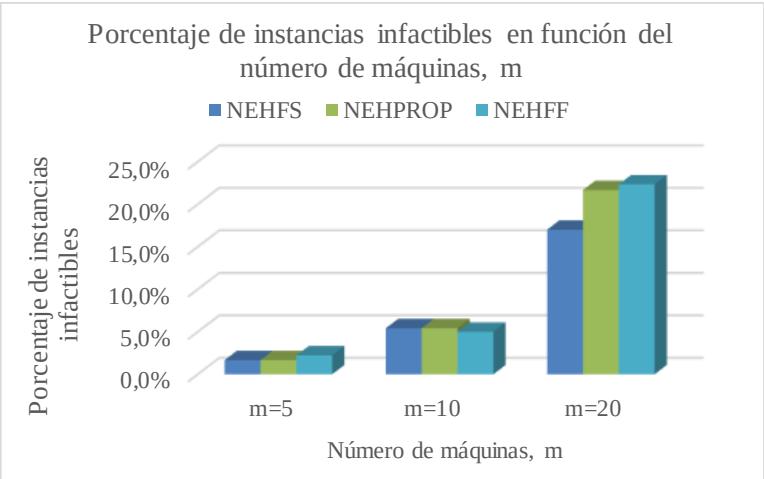

Figura 4.14: Número de instancias sin solución factible en función del número de máquinas, m

En la Figura 4.15 cabría esperar que el comportamiento fuera similar a la gráfica anterior, sin embargo, se observa que la tendencia es totalmente distinta. Lo que se esperaría es que a medida que el número de trabajos aumenta, el número de secuencias infactibles aumentara también, sin embargo, esto no ocurre. Para las instancias comprendidas entre n=20 y n=200, la tendencia es decreciente, aunque los sistemas con menor número de trabajos deberían de ser más fáciles de resolver. La respuesta a esta situación se obtiene de la Función Reparación, apartado 4.1.4. En la función reparación el límite fijado de iteraciones es directamente proporcional al número de trabajos, esto permite que para instancias con una n alta, las posibilidades para encontrar una secuencia factible sean mayores. Sin embargo, se hace evidente que con n=500 la dificultad para que los algoritmos encuentren una solución factible es mucho mayor. Aunque las características de la Función Reparación deberían de facilitar encontrar una secuencia factible para una n tan alta, esto no ocurre porque al tener tal cantidad de trabajos la dificultad para los algoritmos y el tiempo de procesamiento aumenta.

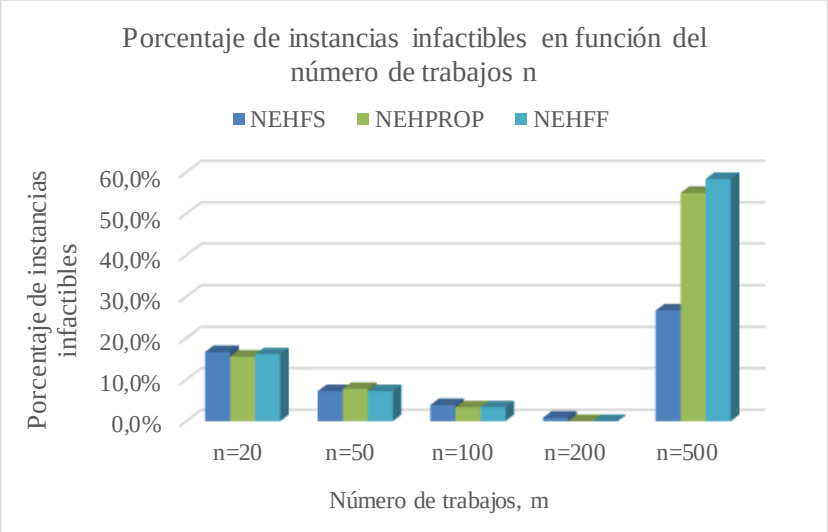

Figura 4.15: Número de instancias sin solución factible en función del número de trabajos, n

5 CONCLUSIONES

El progreso y el desarrollo son imposibles si uno sigue haciendo las cosas tal como siempre las ha hecho.

- Wayne W. Dyer-

En un artículo presentado por la tutora de este proyecto, Paz Pérez González, (Perez-Gonzalez & Framinan, 2014), se comprueba que los problemas con enfoque *ε-constraint* no se pueden resolver de forma polinomial, salvo para casos muy sencillos, incluso para sistemas con una máquina y dos objetivos. Sin embargo, se han propuesto relativamente pocos algoritmos aproximados para estos problemas. Precisamente por esta razón, se ha querido hacer una contribución a este campo abordando un problema multiobjetivo con una restricción *ε-constraint.*

Para abordar el estudio del problema se ha comenzado fijando el valor de épsilon para la restricción ε-*constraint.* Para ello se ha adaptado una heurística constructiva para dos conjuntos de trabajos, con el fin de encontrar un valor de épsilon que facilitara soluciones factibles en la resolución del problema. En segundo lugar, se han propuesto tres heurísticas diferentes para poder analizar y comparar los resultados que proporcionan cada una de ellas. Por tanto, gran peso de este proyecto ha recaído en la programación de las heurísticas adaptadas.

Del análisis de la experimentación computacional se puede concluir que existe una clara diferencia al usar diferentes mecanismos de desempate, es decir, que es un criterio a tener muy en cuenta a la hora de modelar un problema de programación de estas características. Por otra parte, es interesante observar cómo cambia el porcentaje de factibilidad según el valor de α, o lo que es lo mismo, según el valor de épsilon. Como era de esperar a medida que la restricción se va haciendo más limitativa, el porcentaje de factibilidad cae. También se ha observado que para un número de máquinas grande el porcentaje de factibilidad se va reduciendo, sin embargo, a medida que el número de trabajos va en aumento, las heurísticas funcionan mejor y proporcionan más soluciones factibles, salvo para valores muy grandes. Esto se debe a la manera en la que se ha definido la Función Reparación, el número de iteraciones posibles es directamente proporcional al número de trabajos.

Una vez planteadas las conclusiones que se han obtenido en este proyecto es interesante pensar en las posibles aplicaciones de un problema de estas características en la industria actual. Se puede imaginar una industria en la que usándose las mismas máquinas, tengan que hacer frente a dos pedidos diferentes. Póngase como ejemplo una línea de producción de una gran empresa de bebidas, donde las tareas a realizar son la elaboración del jarabe, el agregado de gas carbónico y el envasado y el codificado. Supóngase ahora que existen dos pedidos, uno con una prioridad de entrega alta, conjunto A y otro que tiene una prioridad de entrega menor. Por tanto, el objetivo será minimizar el tiempo de producción del primer pedido sin que el segundo se retrase en exceso. El problema planteado en este proyecto y por tanto su resolución, podría ser aplicable a este caso práctico.

Viendo el alcance de este tipo de problemas en la industria real, sería interesante seguir investigando este tipo de problemas de programación de la producción. Una de las líneas que ya se están investigando y que pueden ser más aplicables, son los problemas de flow-shop con más de dos conjuntos de trabajos. También podría plantearse la resolución de este problema cambiando la restricción del segundo conjunto, como por ejemplo, restringir el Máximum Tardiness. En último lugar sería interesante pensar en modificar la Función Reparación planteada en este proyecto para que no limite tanto a las secuencias que tengan un menor tamaño con respecto al número de trabajos, y analizar las mejoras que supondría para la resolución de este problema. Las posibles combinaciones o resoluciones de este tipo de problemas son muy amplias, pero es una línea de investigación que puede ser muy útil en la práctica.

REFERENCIAS

Bautista Valhondo, J; Llovera Sáez, F. J. (2014). Organización de la Producción: Una perspectiva histórica. In *Organización de la Producción: Una perspectiva histórica* (Vol. 53, Issue 9). https://doi.org/10.1017/CBO9781107415324.004

Code :: Blocks. (n.d.). Retrieved September 6, 2020, from http://www.codeblocks.org/

Dong, X., Huang, H., & Chen, P. (2008). An improved NEH-based heuristic for the permutation flowshop problem. *Computers & Operations Research*, *35*(12), 3962–3968. https://doi.org/10.1016/J.COR.2007.05.005

Fernandez-Viagas, V., & Framinan, J. M. (2014a). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, *45*, 60–67. https://doi.org/https://doi.org/10.1016/j.cor.2013.12.012

Fernandez-Viagas, V., & Framinan, J. M. (2014b). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, *45*, 60–67. https://doi.org/10.1016/J.COR.2013.12.012

Fernandez-Viagas, V., Ruiz, R., & Framinan, J. M. (2017). A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research*, *257*(3), 707–721. https://doi.org/10.1016/j.ejor.2016.09.055

Framinan, J. M., Leisten, R., & Rajendran, C. (2003). Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, *41*(1), 121–148. https://doi.org/10.1080/00207540210161650

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and heuristic in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, *5*, 287–326. https://ac.els-cdn.com/S016750600870356X/1-s2.0-S016750600870356X-main.pdf?_tid=cbf345a3-808d-42df-9560-bf8a52069d0e&acdnat=1550949881_e9837bfdf6316caefaf71ae2f3779b10

Hoogeveen, H. (2005). Multicriteria scheduling. *European Journal of Operational Research*, *167*(3), 592–623. https://doi.org/https://doi.org/10.1016/j.ejor.2004.07.011

Kalczynski, P. J., & Kamburowski, J. (2008). An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers & Operations Research*, *35*(9), 3001–3008. https://doi.org/https://doi.org/10.1016/j.cor.2007.01.020

Mckay, K., Pinedo, M., & Webster, S. (2002). PRACTICE-FOCUSED RESEARCH ISSUES FOR SCHEDULING SYSTEMS * supply chain . A number of authors have discussed the weak connection between results of computer-based scheduling systems by the industrial sector during the past 10 years (Naj areas of supply ch. *Production and Operations Management*, *11*(2), 249–259.

Microsoft Excel, software de hojas de cálculo. (n.d.). Retrieved September 6, 2020, from https://www.microsoft.com/es-es/microsoft-365/excel

Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*(1), 91–95. https://doi.org/10.1016/0305-0483(83)90088-9

Perez-Gonzalez, P., & Framinan, J. M. (2014). A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: Multi-agent scheduling problems. *European Journal of Operational Research*, *235*(1), 1–16. https://doi.org/https://doi.org/10.1016/j.ejor.2013.09.017

Pinedo, M. (1995). *Scheduling : theory, algorithms, and systems* . Prentice Hall.

Pinedo, M. (2012). *Scheduling : theory, algorithms, and systems* (4th ed.). Springer.

Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, *165*(2), 479–494. https://doi.org/10.1016/j.ejor.2004.04.017

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, *47*(1), 65–74. https://doi.org/10.1016/0377-2217(90)90090-X

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278–285. https://doi.org/https://doi.org/10.1016/0377-2217(93)90182-M

GLOSARIO

Máquina: Recurso productivo con capacidad para realizar operaciones de transformación de material.

Conjunto de máquinas $M = \{1, \dots, m\}$

Índices para las máquinas $i \in M$

Trabajo: Producto que es objeto de una operación en alguna de las máquinas del problema a tratar.

Conjunto de trabajos $N = \{1, \dots, n\}$

Índices para los trabajos $j \in N$

Tiempo de proceso (P_{ij}): Es el tiempo en el que la máquina $i \in M$ está ocupada en procesar el trabajo $j \in N$.

Schedule: Asignación en la escala temporal concreta de las máquinas de una empresa para la fabricación de un conjunto de trabajos.

Heurística: Para los problemas combinatorios complejos que requieren esfuerzos computacionales altos, las heurísticas son conjuntos de algoritmos que buscan soluciones lo más óptimas posibles renunciando a encontrar la solución global del problema.

Programa óptimo (optimal schedule): Es el programa admisible más óptimo con respecto al objetivo de estudio.

Algoritmo: Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

Pseudocódigo: es una descripción de alto nivel compacta e informal del principio operativo de un programa informático u otro algoritmo.

Completion time (C_j): Tiempo de terminación de un trabajo j en el conjunto de máquinas del entorno

Makespan: En secuenciación, el objetivo makespan de un proyecto es minimizar el intervalo de tiempo entre el inicio del procesamiento del primer trabajo y el tiempo de finalización del último. $C_{max} = \max_{1 \le j \le n} C_j$

Flowshop o taller de flujo regular: Es un tipo de entorno de fabricación lineal en el que se programa la secuencia de procesamiento de N trabajos en M máquinas. Todos los trabajos tienen que pasar por todas las máquinas en el mismo orden, es decir, teniendo todos los trabajos la misma ruta. $R_j = (1,2 \dots m)\ \forall j \in N$.

NP-HARD: En teoría de la complejidad computacional, la clase de complejidad NP-hard (o NP-complejo, o NP-difícil) es el conjunto de los problemas de decisión que contiene los problemas no polinomiales para los que no se puede hallar a priori una solución óptima.

ANEXOS

En los siguientes anexos, se detallan los códigos en lenguaje C++ construidos y utilizados para el desarrollo de este Proyecto. Estos códigos han sido desarrollados en el entorno de desarrollo integrado (IDE) llamado Code::Blocks. Estos anexos recogen todos aquellos algoritmos necesarios no solo para resolver el problema de este Proyecto, sino también para encontrar valores auxiliares o que se necesitaban con antelación para resolver el problema. También se encontrará un código que contiene una función en C que se utilizará como Función Reparación dentro de los algoritmos adaptados de NEH.

Anexo A

Código algoritmo NEH adaptado a dos grupos de trabajos para generar épsilon

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char*argv[])
{
/*ESTIMACIÓN VALOR INICIAL DE EPSILON*/

 printf("VAMOS A USAR EL METODO NEH PARA ESTUDIAR LOS DOS GRUPOS DE TRABAJOS\n");

/*FUNCIÓN LECTURA DE FICHEROS*/

FILE *archivo;
int temporal;
MAT_LONG pt;
int n=0;
int m=0;
int pi=0;
int pj=0;

archivo = fopen(argv[1],"rt");
if (archivo == NULL)
      {
         printf("\n Error de apertura del archivo \n\n");
```

```
    }
    else
    {
       printf("\n El contenido del archivo de prueba es \n\n");
       fscanf(archivo,"%d\n", &temporal);
       m=temporal;
       printf("el n de maquinas es %d\n",m);

       fscanf(archivo,"%d\n", &temporal);
       n=temporal;
       printf("el n de trabajos es %d\n",n);
       pt=DIM_MAT_LONG(n,m);

       for(pi=0;pi<n;pi++)
       {
       for(pj=0;pj<m;pj++)
          {
          fscanf(archivo,"%10d", &temporal);
          pt[pi][pj]=temporal;
          }
       }
    }
    fclose(archivo);

/*PARA ESTE PROBLEMA SE TOMARÁ EL TOTAL DE LOS TIEMPOS DE PROCESO DE LOS
TRABAJOS EN TODAS LAS MAQUINAS*/

   VECTOR_LONG TIEMPOS_TOTALES=DIM_VECTOR_LONG(n);
   int i=0;
   int j=0;
   for(i=0;i<(n);i++)
   {
      TIEMPOS_TOTALES[i]=0;
   }

   for(i=0;i<(n);i++)
   {
      for(j=0;j<(m);j++)
```

```
        {
           TIEMPOS_TOTALES[i]=TIEMPOS_TOTALES[i]+pt[i][j];
        }
    }
    /*print_vector(TIEMPOS_TOTALES,n);*/

/*CON LOS TIEMPOS TOTALES DE TODOS LOS TRABAJOS SE SELECCIONAN LOS TRABAJOS
DEL GRUPO A*/

    VECTOR_LONG GRUPO_A=DIM_VECTOR_LONG(n/2);
    for(i=0;i<(n/2);i++)
    {
       GRUPO_A[i]=TIEMPOS_TOTALES[i];
    }
    /*print_vector(GRUPO_A,n/2);*/

/*SE ORDENA SEGÚN LA REGLA LPT ÚNICAMENTE LOS CORRESPONDIENTES AL GRUPO A*/

    VECTOR_INT LPT_A=DIM_VECTOR_INT(n/2);
    VECTOR_LONG vector_copia=DIM_VECTOR_LONG(n/2);
    copy_vector(GRUPO_A,vector_copia,n/2);
    sort_vector(vector_copia,LPT_A,n/2,'D');
   /*printf("EL VECTOR ORDENADO DEL GRUPO A ES");
    print_vector(LPT_A,n/2);*/

/*CONSTRUCCIÓN DE LAS DOS PRIMERAS COMBINACIONES A EVALUAR*/
    int aux=2;
    VECTOR_INT COMIENZO_A=DIM_VECTOR_INT(aux);
    VECTOR_INT ALTERNATIVO_A=DIM_VECTOR_INT(aux);
    for(i=0;i<aux;i++)
       {
         COMIENZO_A[i]=LPT_A[i];
       }
    print_vector(COMIENZO_A,aux);
    ALTERNATIVO_A[0]=COMIENZO_A[1];
    ALTERNATIVO_A[1]=COMIENZO_A[0];

/*CONSTRUCCIÓN DEL MAKESPAN*/
```

```
long int cm=0;
long int cmx_1=0;
long int cmx_11=0;
cmx_1=pt[COMIENZO_A[0]][0];
cmx_11=cmx_1+pt[COMIENZO_A[1]][0];
for(j=1;j<m;j++){
        cmx_1=cmx_1+pt[COMIENZO_A[0]][j];
        if(cmx_1>=cmx_11)
            {
               cm=cmx_1+pt[COMIENZO_A[1]][j];
            }else
            {
               cm=cmx_11+pt[COMIENZO_A[1]][j];
            }
        cmx_11=cm;
    }
printf("makespan de la primera opcion es %ld\n",cm);
cm=0;
long int cmx_2=0;
long int cmx_21=0;
cmx_2=cmx_2+pt[ALTERNATIVO_A[0]][0];
cmx_21=cmx_2+pt[ALTERNATIVO_A[1]][0];
for(j=1;j<m;j++)
    {
        cmx_2=cmx_2+pt[ALTERNATIVO_A[0]][j];
         if(cmx_2>=cmx_21)
            {
               cm=cmx_2+pt[ALTERNATIVO_A[1]][j];
            }else
            {
               cm=cmx_21+pt[ALTERNATIVO_A[1]][j];
            }
        cmx_21=cm;
    }
printf("makespan de la segunda opcion es %ld\n",cmx_21);
```

/* SE EVALUA EL MAKESPAN PARA LAS DOS PRIMERAS SECUENCIAS PARCIALES POSIBLES*/

```
   VECTOR_INT secuencia_inicio=DIM_VECTOR_INT(n);
   if(cmx_11<cmx_21){
         copy_vector(COMIENZO_A,secuencia_inicio, aux);
   }else{
         copy_vector(ALTERNATIVO_A,secuencia_inicio,aux);
   }
  print_vector(secuencia_inicio,aux);

/*ITERACIONES DEL ALGORITMO NEH PARA EL GRUPO A DE TRABAJOS*/

   int w=0;
   int p=0;
   int r=0;
   int c=0;
   long int makesp1=0;
   long int makesp2=500000000;
   for(c=3;c<=n/2;c++)
   {
      makesp2=500000000;
      VECTOR_LONG TIEMPO=DIM_VECTOR_LONG(m);
      VECTOR_INT sec_d=DIM_VECTOR_INT(c);
      for(p=0;p<c;p++)
      {
         /*INSERTAR EL NUEVO TRABAJO EN LAS DIFERENTES POSICIONES*/
         insert_vector(secuencia_inicio,c,LPT_A[c-1], p);

         long int sum=0;
         for(r=0;r<m;r++)
            {
            TIEMPO[r]=sum+pt[secuencia_inicio[0]][r];
            sum=TIEMPO[r];
            }
         /*BUCLE PARA BUSCAR EL MENOR MAKESPAN*/
         for(i=1;i<c;i++)
            {
               TIEMPO[0]=TIEMPO[0]+pt[secuencia_inicio[i]][0];

               for(w=1; w<m; w++)
```

```
            {   /*printf("m ");*/
                if (TIEMPO[w]>=TIEMPO[w-1])
                {
                    TIEMPO[w]=TIEMPO[w]+pt[secuencia_inicio[i]][w];
                }
                if(TIEMPO[w]<TIEMPO[w-1])
                {
                    TIEMPO[w]=TIEMPO[w-1]+pt[secuencia_inicio[i]][w];
                }
                print_vector(TIEMPO,m);
            }
        }
        makesp1=TIEMPO[m-1];
        printf("el makespan para esta secuencia es: %ld\n",makesp1);
        if(makesp1<makesp2)
        {
            copy_vector(secuencia_inicio,sec_d,c);
            makesp2=makesp1;
        }
        extract_vector(secuencia_inicio,c,p);
        print_vector(secuencia_inicio,c-1);

    }

    copy_vector(sec_d,secuencia_inicio,c);
    printf("la secuencia para la siguiente iteracion es:\n");
    print_vector(secuencia_inicio,c);
}
printf("el makespan final para esta secuencia es: %ld\n",makesp2);



/*SEGUNDA PARTE: GRUPO B (LOS TRABAJOS DEL GRUPO B SON LOS PERTENECIENTES A LA
SEGUNDA MITAD DE LOS TRABAJOS)*/


/*SE ORDENAN LOS TRABAJOS SEGÚN LA REGLA LPT*/
VECTOR_LONG GRUPO_B=DIM_VECTOR_LONG(n/2);
for(i=0;i<(n/2);i++)
{
```

```
    GRUPO_B[i]=TIEMPOS_TOTALES[(n/2)+i];
  }

  VECTOR_INT LPT_B=DIM_VECTOR_INT(n);
  VECTOR_LONG vector_copia_b=DIM_VECTOR_LONG(n/2);
  copy_vector(GRUPO_B,vector_copia_b,n/2);
  sort_vector(vector_copia_b,LPT_B,n/2,'D');
  printf("EL VECTOR ORDENADO DEL GRUPO B ES");
  print_vector(LPT_B,n/2);

/*PARA PODER HACER USO DEL VECTOR TOTAL DE TIEMPOS ES NECESARIO QUE LAS
POSICIONES COINCIDAN CON LAS DE EL VECTOR TOTAL, NO LA DEL VECTOR PARCIAL*/
  VECTOR_INT LPT_B_total=DIM_VECTOR_INT(n/2);
  for(i=0;i<(n/2);i++)
  {
    LPT_B_total[i]=LPT_B[i]+n/2;
  }

/*APLICACIÓN DEL ALGORITMO NEH PARA EL GRUPO B*/
  int wb=0;
  int pb=0;
  int rb=0;
  int jb=0;
  unsigned long int makesp1b=0;
  unsigned long int makesp2b=500000000;
  for(jb=((n/2)+1);jb<=n;jb++)
  {  makesp2b=500000000;
    VECTOR_LONG TIEMPO=DIM_VECTOR_LONG(m);
    VECTOR_INT sec_d=DIM_VECTOR_INT(jb);
    /* printf("next ");*/
    for(pb=n/2;pb<jb;pb++)
    {
      /*INSERTAR EL NUEVO TRABAJO EN LAS DIFERENTES POSICIONES*/
      insert_vector(secuencia_inicio,jb,LPT_B_total[jb-1-(n/2)], pb);

      long int sum=0;
      for(rb=0;rb<m;rb++)
        {
```

```
            TIEMPO[rb]=sum+pt[secuencia_inicio[0]][rb];
            sum=TIEMPO[rb];
            }

         /*BUCLE PARA BUSCAR EL MENOR MAKESPAN*/
         for(i=1;i<jb;i++)
            {
               TIEMPO[0]=TIEMPO[0]+pt[secuencia_inicio[i]][0];
               for(wb=1;wb<m;wb++)
               {
                  if (TIEMPO[wb]>TIEMPO[wb-1])
                  {
                     TIEMPO[wb]=TIEMPO[wb]+pt[secuencia_inicio[i]][wb];
                  }else{
                     TIEMPO[wb]=TIEMPO[wb-1]+pt[secuencia_inicio[i]][wb];
                  }
               }
            }
         makesp1b=TIEMPO[m-1];
         printf("el makespan para esta secuencia es: %ld\n",makesp1b);
         if(makesp1b<makesp2b)
         {
            copy_vector(secuencia_inicio,sec_d,jb);
            makesp2b=makesp1b;
         }
         extract_vector(secuencia_inicio,jb,pb);

      }
       copy_vector(sec_d,secuencia_inicio,jb);
       printf("la secuencia para la siguiente iteracion es:\n");
       print_vector(secuencia_inicio,jb);
   }
   printf("el MAKESPAN para esta secuencia es: %ld\n",makesp2b);

/*GUARDAR DATOS DE SALIDA*/
   FILE* fichero;
   fichero = fopen("salidaepsilon.txt", "a");
   fprintf (fichero, "%ld\n", makesp2b, '\n');
```

```
    fclose(fichero);
    printf("Proceso completado");

/*LIBERAR MEMORIA*/
free_mat_long(pt,n);
free(secuencia_inicio);
free(TIEMPOS_TOTALES);
free(LPT_A);
free(LPT_B);
free(LPT_B_total);
free(GRUPO_A);
free(GRUPO_B);
free(vector_copia);
free(vector_copia_b);
free(COMIENZO_A);
free(ALTERNATIVO_A);

    return 0;
}
```

Anexo B

Código algoritmo NEHFF aplicado a dos grupos de trabajos para generar épsilon

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
int main(int argc, char*argv[])
{
/*FUNCIÓN LECTURA DE FICHEROS*/

FILE *archivo;
int temporal;
MAT_LONG pt;
int n=0;
int m=0;
archivo = fopen(argv[1],"rt");

if (archivo == NULL)
        {
           printf("\n Error de apertura del archivo \n\n");
        }
        else
        {
           printf("\n El contenido del archivo de prueba es \n\n");
           fscanf(archivo,"%d\n", &temporal);
           m=temporal;
           printf("el n de maquinas es %d\n",m);

           fscanf(archivo,"%d\n", &temporal);
           n=temporal;
           printf("el n de trabajos es %d\n",n);
           pt=DIM_MAT_LONG(n,m);

           for(pi=0;pi<n;pi++)
           {
           for(pj=0;pj<m;pj++)
```

```
            {
            fscanf(archivo,"%10d", &temporal);
            pt[pi][pj]=temporal;
            }
        }
    }
    fclose(archivo);

/*PARA ESTE PROBLEMA SE TOMARÁ EL PROMEDIO DE LOS TIEMPOS DE PROCESO DE LOS
TRABAJOS EN TODAS LAS MÁQUINAS*/

  VECTOR_DOUBLE TIEMPOS_PROMEDIO=DIM_VECTOR_DOUBLE(n);
  int i=0;
  int j=0;
  for(i=0;i<(n);i++)
  {
    TIEMPOS_PROMEDIO[i]=0;
  }

  for(i=0;i<(n);i++)
  {
    for(j=0;j<(m);j++)
        {
          TIEMPOS_PROMEDIO[i]=TIEMPOS_PROMEDIO[i]+pt[i][j];
        }
  }
  for(i=0;i<(n);i++)
  {
    TIEMPOS_PROMEDIO[i]=TIEMPOS_PROMEDIO[i]/m;
  }

/*DESVIACION ESTANDAR DE LOS TIEMPOS DE PROCESO*/

  VECTOR_DOUBLE DESVIACION=DIM_VECTOR_DOUBLE(n);
  VECTOR_DOUBLE SUMATORIO=DIM_VECTOR_DOUBLE(n);
  for(i=0;i<(n);i++)
  {
    SUMATORIO[i]=0;
```

```
    }

    for(i=0;i<(n);i++)
    {
        for(j=0;j<(m);j++)
            {
             SUMATORIO[i]=SUMATORIO[i]+((pt[i][j]-TIEMPOS_PROMEDIO[i])*(pt[i][j]-
TIEMPOS_PROMEDIO[i]));
            }
    }
    for(i=0;i<(n);i++)
    {
        SUMATORIO[i]=SUMATORIO[i]/(m-1);
    }
    for(i=0;i<(n);i++)
    {
        DESVIACION[i]=sqrt(SUMATORIO[i]);
    }

/*TOTAL SECUENCIA INICIAL AvgDev*/

    VECTOR_FLOAT SEC=DIM_VECTOR_FLOAT(n);
    for(i=0;i<(n);i++)
    {
        SEC[i]=TIEMPOS_PROMEDIO[i]+DESVIACION[i];
    }
/*DIVIDIMOS EN EL GRUPO A Y EN EL GRUPO B*/
    VECTOR_FLOAT GRUPO_A=DIM_VECTOR_FLOAT(n/2);
    for(i=0;i<(n/2);i++)
    {
        GRUPO_A[i]=SEC[i];
    }
    VECTOR_FLOAT GRUPO_B=DIM_VECTOR_FLOAT(n/2);
    for(i=0;i<(n/2);i++)
    {
        GRUPO_B[i]=SEC[(n/2)+i];
    }
/*ORDENAMOS DE FORMA DESCENDENTE SEGÚN AvgDev TANTO EL GRUPO A COMO EL
```

```
GRUPO B*/

   VECTOR_INT LPT_A=DIM_VECTOR_INT(n/2);
   VECTOR_FLOAT vector_copia=DIM_VECTOR_FLOAT(n/2);
   copy_vector(GRUPO_A,vector_copia,n/2);
   sort_vector(vector_copia,LPT_A,n/2,'D');

   VECTOR_INT LPT_B=DIM_VECTOR_INT(n/2);
   VECTOR_FLOAT vector_copiab=DIM_VECTOR_FLOAT(n/2);
   copy_vector(GRUPO_B,vector_copiab,n/2);
   sort_vector(vector_copiab,LPT_B,n/2,'D');

/*PARA PODER HACER USO DEL VECTOR TOTAL DE TIEMPOS ES NECESARIO QUE LAS
POSICIONES COINCIDAN CON LAS DE EL VECTOR TOTAL, NO LA DEL VECTOR PARCIAL*/

   VECTOR_INT LPT_B_total=DIM_VECTOR_INT(n/2);
   for(i=0;i<(n/2);i++)
   {
      LPT_B_total[i]=LPT_B[i]+n/2;
   }

/*CONSTRUCCIÓN DE LAS DOS PRIMERAS COMBINACIONES A EVALUAR*/
   int aux=2;
   VECTOR_INT COMIENZO_A=DIM_VECTOR_INT(aux);
   VECTOR_INT ALTERNATIVO_A=DIM_VECTOR_INT(aux);
   for(i=0;i<aux;i++)
      {
        COMIENZO_A[i]=LPT_A[i];
      }
   print_vector(COMIENZO_A,aux);
   ALTERNATIVO_A[0]=COMIENZO_A[1];
   ALTERNATIVO_A[1]=COMIENZO_A[0];
   print_vector(ALTERNATIVO_A,aux);

/*CONSTRUCCIÓN DEL MAKESPAN*/
/*EVALUAMOS LA PRIMERA DE LAS OPCIONES*/
   VECTOR_LONG it1=DIM_VECTOR_LONG(n);
   it1[0]=0;
```

```
    long int it1t=0;
    long int cm=0;
    long int cmx_1=0;
    long int cmx_11=0;
    cmx_1=pt[COMIENZO_A[0]][0];
    cmx_11=cmx_1+pt[COMIENZO_A[1]][0];
    for(j=1;j<m;j++){
          cmx_1=cmx_1+pt[COMIENZO_A[0]][j];
          if(cmx_1>=cmx_11)
             {
                cm=cmx_1+pt[COMIENZO_A[1]][j];
             }else
             {
                cm=cmx_11+pt[COMIENZO_A[1]][j];
             }
          cmx_11=cm;
          it1[j]=cm-pt[COMIENZO_A[0]][j]-pt[COMIENZO_A[1]][j];
          it1t=it1t+it1[j];
       }

/*EVALUAMOS LA SEGUNDA OPCIÓN*/
    VECTOR_LONG it2=DIM_VECTOR_LONG(n);
    it2[0]=0;
    long int it2t=0;
    cm=0;
    long int cmx_2=0;
    long int cmx_21=0;
    cmx_2=cmx_2+pt[ALTERNATIVO_A[0]][0];
    cmx_21=cmx_2+pt[ALTERNATIVO_A[1]][0];
    for(j=1;j<m;j++)
       {
          cmx_2=cmx_2+pt[ALTERNATIVO_A[0]][j];
           if(cmx_2>=cmx_21)
             {
                cm=cmx_2+pt[ALTERNATIVO_A[1]][j];
             }else
             {
                cm=cmx_21+pt[ALTERNATIVO_A[1]][j];
```

```
            }
        cmx_21=cm;
        it2[j]=cm-pt[ALTERNATIVO_A[0]][j]-pt[ALTERNATIVO_A[1]][j];
        it2t=it2t+it2[j];
      }

/* SE EVALUA EL MAKESPAN PARA LAS DOS PRIMERAS SECUENCIAS PARCIALES POSIBLES*/

   VECTOR_INT secuencia_inicio=DIM_VECTOR_INT(n);
   if(cmx_11<cmx_21){
        copy_vector(COMIENZO_A,secuencia_inicio, aux);
   }
   if(cmx_11>cmx_21){
        copy_vector(ALTERNATIVO_A,secuencia_inicio,aux);
   }
   if(cmx_11==cmx_21){
        if(it1t<it2t){
            copy_vector(ALTERNATIVO_A,secuencia_inicio,aux);
        }else{
           copy_vector(ALTERNATIVO_A,secuencia_inicio,aux);
        }
   }
VECTOR_INT auxiliar=DIM_VECTOR_INT(n);
/*ITERACIONES DEL ALGORITMO NEH PARA EL GRUPO A DE TRABAJOS*/

   long int itA=0;
   long int itAt=0;
   long int it=0;
   int w=0;
   int p=0;
   int r=0;
   int e=0;
   int l=0;
   int c=0;
   long int makesp1=0;
   long int makesp2=500000000;
   for(c=3;c<=n/2;c++)
   {   MAT_LONG E=DIM_MAT_LONG(c,m+1);
```

```
VECTOR_INT SECUENCIA=DIM_VECTOR_INT(c);
copy_vector(secuencia_inicio,SECUENCIA,c-1);

/*CÁLCULO DE E para k-1*/
   for(r=0;r<=m;r++)
   {
      E[0][r]=0;
   }
   for(r=0;r<=c-1;r++)
   {
      E[r][0]=0;
   }
   for(r=1;r<=m;r++)
      {
      for(e=1;e<=c-1;e++)
      {
         E[e][r]=Max(E[e-1][r],E[e][r-1])+pt[secuencia_inicio[e-1]][r-1];
      }
      }
   /*CÁLCULO DE Q para k-1*/
   MAT_LONG Q=DIM_MAT_LONG(c,m+1);
   for(r=0;r<=m;r++)
   {
      Q[c-1][r]=0;
   }
   for(r=0;r<=c-1;r++)
   {
      Q[r][m]=0;
   }
   for(r=m-1;r>=0;r--)
      {
      for(e=c-2;e>=0;e--)
      {
         Q[e][r]=Max(Q[e][r+1],Q[e+1][r])+pt[secuencia_inicio[e]][r];
      }
      }
   /*CÁLCULO DE F para i=1...m*/
   MAT_LONG F=DIM_MAT_LONG(c,m+1);
```

```
        for(r=0;r<=c-1;r++)
        {
           F[r][0]=0;
        }
        for(r=1;r<=m;r++)
           {
           for(l=0;l<c;l++)
           {
              F[l][r]=Max(E[l][r],F[l][r-1])+pt[LPT_A[c-1]][r-1];
           }
           }
//         print_double_matrix(F,c,m+1);
//         printf("next F\n ");

      /*PARTE ANTERIOR PARA COMENZAR EL ALGORITMO*/
      int tb=0;
      VECTOR_LONG MAKESPANES=DIM_VECTOR_LONG(c);
      makesp2=500000000;
      VECTOR_LONG TIEMPO=DIM_VECTOR_LONG(m);
      VECTOR_INT sec_d=DIM_VECTOR_INT(c);
      for(p=0;p<c;p++)
      {
         /*INSERTAR EL NUEVO TRABAJO EN LAS DIFERENTES POSICIONES*/
         insert_vector(secuencia_inicio,c,LPT_A[c-1], p);

         long int sum=0;
         for(r=0;r<m;r++)
           {
           TIEMPO[r]=sum+pt[secuencia_inicio[0]][r];
           sum=TIEMPO[r];
           }
         /*BUCLE PARA BUSCAR EL MENOR MAKESPAN*/
         for(i=1;i<c;i++)
           {
              TIEMPO[0]=TIEMPO[0]+pt[secuencia_inicio[i]][0];

              for(w=1; w<m; w++)
              {
```

```
            if (TIEMPO[w]>=TIEMPO[w-1])
            {
               TIEMPO[w]=TIEMPO[w]+pt[secuencia_inicio[i]][w];
            }
            if(TIEMPO[w]<TIEMPO[w-1])
            {
               TIEMPO[w]=TIEMPO[w-1]+pt[secuencia_inicio[i]][w];
            }
         }
      }
   makesp1=TIEMPO[m-1];
   MAKESPANES[p]=makesp1;
   for(w=1; w<m; w++)
   {  for (aux=0;aux<c;aux++)
      {
         itA=itA+pt[aux][w];
      }
     itAt=itAt+(TIEMPO[w]-itA);
     itA=0;
   }
   if(makesp1<makesp2)
   {
      copy_vector(secuencia_inicio,sec_d,c);
      makesp2=makesp1;
      it=itAt;
   }
   if(makesp1=makesp2)
   {
      if(itAt<=it){
         copy_vector(secuencia_inicio,sec_d,c);
         makesp2=makesp1;
         it=itAt;}
   }
   extract_vector(secuencia_inicio,c,p);
   itA=0;
   itAt=0;

}
```

```
/*CALCULO DE LAS 3 VARIABLES NECESARIAS PARA DESARROLLAR EL PROBLEMA*/
long int minimo=MAKESPANES[0];
for(i=1;i<c;i++)
   {
      if(MAKESPANES[i]<minimo)
      {
        minimo=MAKESPANES[i];
      }
   }
for(i=0;i<c;i++)
   {
      if(MAKESPANES[i]==minimo)
      {
        tb=tb+1;
      }
   }
VECTOR_INT ptb=DIM_VECTOR_INT(tb);
int y=0;
for(i=0;i<c;i++)
   {
      if(MAKESPANES[i]==minimo)
      {
        ptb[y]=i;
        y=y+1;
      }
   }
 int primero=search_vector(MAKESPANES,c,minimo);

/*DESARROLLO DEL ALGORITMO DE NEHFF*/
long int itbp=500000000;
if(tb>1 && c<n/2)
   {   for(l=0;l<tb;l++)
      {
         long int itprima=0;
         if(ptb[l]==c-1)
         {
            for(i=2;i<=m;i++)
            {
```

```
                    itprima=itprima+F[c-1][i]-E[c-1][i]-pt[LPT_A[c-1]][i-1];
                }
            }else{
                MAT_DOUBLE Fprima=DIM_MAT_DOUBLE(c,m+1);
                Fprima[ptb[l]][1]=F[ptb[l]][1]+pt[ptb[l]][0];
                for(i=2;i<=m;i++)
                {
                itprima=itprima+F[ptb[l]][i]-E[ptb[l]+1][i]+pt[secuencia_inicio[ptb[l]]][i-1]-pt[LPT_A[c-1]][i-1]+Max(0,Fprima[ptb[l]][i-1]-F[ptb[l]][i]);
                    Fprima[ptb[l]][i]=Max(Fprima[ptb[l]][i-1],F[ptb[l]][i])+pt[ptb[l]][i-1];
                }
            }
            if(itbp>=itprima)
            {
                primero=ptb[l];
                itbp=itprima;
            }
        }
    }
    insert_vector(SECUENCIA,c,LPT_A[c-1], primero);
    copy_vector(SECUENCIA,secuencia_inicio,c);
    if(c==n/2){
        copy_vector(SECUENCIA,secuencia_inicio,c);}
    tb=0;
}
printf("el TI de A para esta secuencia es: %ld\n",it);

/*SEGUNDA PARTE: GRUPO B (LOS TRABAJOS DEL GRUPO B SON LOS PERTENECIENTES A LA SEGUNDA MITAD DE LOS TRABAJOS)*/

/*APLICACIÓN DEL ALGORITMO NEH PARA EL GRUPO B*/
long int itB=0;
long int itBt=0;
long int ibt=0;
int wb=0;
int pb=0;
int rb=0;
int jb=0;
```

```
unsigned long int makesp1b=0;
unsigned long int makesp2b=500000000;
for(jb=((n/2)+1);jb<=n;jb++)
{   MAT_DOUBLE E=DIM_MAT_DOUBLE(jb,m+1);
    VECTOR_INT SECUENCIA=DIM_VECTOR_INT(jb);
    copy_vector(secuencia_inicio,SECUENCIA,jb-1);
    /*CÁLCULO DE E para k-1*/
        for(r=0;r<=m;r++)
        {
           E[0][r]=0;
        }
        for(r=0;r<=jb-1;r++)
        {
           E[r][0]=0;
        }
        for(r=1;r<=m;r++)
           {
           for(e=1;e<=jb-1;e++)
           {
              E[e][r]=Max(E[e-1][r],E[e][r-1])+pt[secuencia_inicio[e-1]][r-1];
           }
           }

        /*CÁLCULO DE Q*/
        MAT_DOUBLE Q=DIM_MAT_DOUBLE(jb,m+1);
        for(r=0;r<=m;r++)
        {
           Q[jb-1][r]=0;
        }
        for(r=0;r<=jb-1;r++)
        {
           Q[r][m]=0;
        }
        for(r=m-1;r>=0;r--)
           {
           for(e=jb-2;e>=0;e--)
           {
              Q[e][r]=Max(Q[e][r+1],Q[e+1][r])+pt[secuencia_inicio[e]][r];
```

```
    }
    }

  /*CÁLCULO DE F*/
  MAT_DOUBLE F=DIM_MAT_DOUBLE(jb,m+1);
  for(r=0;r<=jb-1;r++)
  {
    F[r][0]=0;
  }
  for(r=1;r<=m;r++)
    {
    for(l=0;l<jb;l++)
    {
      F[l][r]=Max(E[l][r],F[l][r-1])+pt[LPT_B_total[jb-1-(n/2)]][r-1];
    }
    }

/*PARTE ANTERIOR PARA COPMENZAR A EVALUAR EL MAKESPAN DEL GRUPO B*/
int tbb=0;
VECTOR_LONG MAKESPANESb=DIM_VECTOR_LONG(jb+1-(n/2));
makesp2b=500000000;
VECTOR_LONG TIEMPO=DIM_VECTOR_LONG(m);
VECTOR_INT sec_d=DIM_VECTOR_INT(jb);
for(pb=n/2;pb<jb;pb++)
{
  /*INSERTAR EL NUEVO TRABAJO EN LAS DIFERENTES POSICIONES*/
  insert_vector(secuencia_inicio,jb,LPT_B_total[jb-1-(n/2)], pb);
  long int sum=0;
  for(rb=0;rb<m;rb++)
    {
    TIEMPO[rb]=sum+pt[secuencia_inicio[0]][rb];
    sum=TIEMPO[rb];
    }

  /*BUCLE PARA BUSCAR EL MENOR MAKESPAN*/
  for(i=1;i<jb;i++)
    {
      TIEMPO[0]=TIEMPO[0]+pt[secuencia_inicio[i]][0];
```

```
        for(wb=1;wb<m;wb++)
        {
           if (TIEMPO[wb]>TIEMPO[wb-1])
           {
              TIEMPO[wb]=TIEMPO[wb]+pt[secuencia_inicio[i]][wb];
           }else{
              TIEMPO[wb]=TIEMPO[wb-1]+pt[secuencia_inicio[i]][wb];
           }
        }
     }
makesp1b=TIEMPO[m-1];
MAKESPANESb[pb-n/2]=makesp1b;
for(w=1; w<m; w++)
{
    for (aux=0;aux<jb;aux++)
    {
     itB=itB+pt[aux][w];
    }
   itBt=itBt+(TIEMPO[w]-itB);
   itB=0;
}
if(makesp1b<makesp2b)
{
    copy_vector(secuencia_inicio,sec_d,jb);
    makesp2b=makesp1b;
    ibt=itBt;
}
if(makesp1b==makesp2b)
{
    if(itBt<=ibt){
        copy_vector(secuencia_inicio,sec_d,jb);
        makesp2b=makesp1b;
        ibt=itBt;
    }
}
extract_vector(secuencia_inicio,jb,pb);
itB=0;
itBt=0;
```

```
}
/*CALCULO DE LAS 3 VARIABLES NECESARIAS PARA DESARROLLAR EL PROBLEMA*/
long int minimo=MAKESPANESb[0];
for(i=1;i<jb-(n/2);i++)
   {
      if(MAKESPANESb[i]<minimo)
      {
        minimo=MAKESPANESb[i];
      }
   }
for(i=0;i<jb-n/2;i++)
   {
      if(MAKESPANESb[i]==minimo)
      {
        tbb=tbb+1;
      }
   }
VECTOR_INT ptb=DIM_VECTOR_INT(tbb+1);
int y=0;
for(i=0;i<jb-(n/2);i++)
   {
      if(MAKESPANESb[i]==minimo)
      {  ptb[y]=i;
        y=y+1;
      }
   }
int primero=search_vector(MAKESPANESb,jb-(n/2),minimo);

/*DESARROLLO DEL ALGORITMO DE NEHFF*/
long int itbp=500000000;
if(tbb>1 && jb<n)
   {
      for(l=0;l<tbb;l++)
      {
         long int itprima=0;
         if(ptb[l]==jb-1)
         {
            for(i=2;i<=m;i++)
```

```
                {
                    itprima=itprima+F[jb-1][i]-E[jb-1][i]-pt[LPT_B_total[jb-1-(n/2)]][i-1];
                }
            }else{
                MAT_DOUBLE Fprima=DIM_MAT_DOUBLE(jb,m+1);
                Fprima[ptb[l]][1]=F[ptb[l]][1]+pt[ptb[l]][0];
                for(i=2;i<=m;i++){
                itprima=itprima+F[ptb[l]][i]-E[ptb[l]+1][i]+pt[secuencia_inicio[ptb[l]]][i-1]-
                pt[LPT_B_total[jb-1-(n/2)]][i-1]+Max(0,Fprima[ptb[l]][i-1]-F[ptb[l]][i]);
                    Fprima[ptb[l]][i]=Max(Fprima[ptb[l]][i-1],F[ptb[l]][i])+pt[ptb[l]][i-1];
                }
                }

            if(itbp>=itprima)
            {
                primero=ptb[l];
                makesp2b=MAKESPANESb[ptb[l]];
                itbp=itprima;
            }
          }
        }
        insert_vector(SECUENCIA,jb,LPT_B_total[jb-1-(n/2)], primero+n/2);
        copy_vector(SECUENCIA,secuencia_inicio,jb);
        tbb=0;
        if(jb==n){copy_vector(SECUENCIA,secuencia_inicio,jb);}

    }
   printf("el CMAX FINAL para esta secuencia es: %ld\n",makesp2b);

/*GUARDAR DATOS DE SALIDA*/
   FILE* fichero;
   fichero = fopen("salidaNEHFF.txt", "a");
   fprintf (fichero, "%ld\n", makesp2b, '\n');
   fclose(fichero);
   printf("Proceso completado");


/*LIBERAR MEMORIA*/
```

```
free_mat_long(pt,n);
free(secuencia_inicio);
free(TIEMPOS_PROMEDIO);
free(SUMATORIO);
free(DESVIACION);
free(SEC);
free(LPT_A);
free(LPT_B);
free(LPT_B_total);
free(GRUPO_A);
free(GRUPO_B);
free(vector_copia);
free(vector_copiab);
free(COMIENZO_A);
free(ALTERNATIVO_A);
free(auxiliar);



    return 0;
}
```

Anexo C

Código algoritmo NEH adaptado a dos grupos de trabajos con regla de desempate FS, NEH_{FS}

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
VECTOR_INT funcionreparacion(VECTOR_INT repeticiones, VECTOR_INT secuencia_inicio, VECTOR_INT valoresB_S, VECTOR_INT posicionesB_S, int k, int c, int n, int m, MAT_LONG pt, long int epsilonTOTAL, VECTOR_INT LPT);
int main(int argc, char*argv[])
{
/*ESTIMACIÓN VALOR INICIAL DE EPSILON*/

  printf("VAMOS A USAR EL METODO NEH PARA MODELAR LOS DOS GRUPOS DE TRABAJOS\n");

/*FUNCIÓN LECTURA DE FICHEROS*/

FILE *archivo;
int temporal;
MAT_LONG pt;
int n=0;
int m=0;
int pi=0;
int pj=0;
int epsilon=0;

archivo = fopen(argv[1],"rt");
if (archivo == NULL)
      {
         printf("\n Error de apertura del archivo \n\n");
      }
      else
      {
         printf("\n El contenido del archivo de prueba es \n\n");
         fscanf(archivo,"%d\n", &temporal);
         m=temporal;
```

```
        printf("el n de maquinas es %d\n",m);

        fscanf(archivo,"%d\n", &temporal);
        n=temporal;
        printf("el n de trabajos es %d\n",n);

        fscanf(archivo,"%d\n", &temporal);
        epsilon=temporal;
        printf("el epsilon es %d\n",epsilon);
        pt=DIM_MAT_LONG(n,m);

        for(pi=0;pi<n;pi++)
        {
        for(pj=0;pj<m;pj++)
            {
            fscanf(archivo,"%10d", &temporal);
            pt[pi][pj]=temporal;
            }
        }

    }

    fclose(archivo);

/*PARA ESTE PROBLEMA SE TOMARÁ EL TOTAL DE LOS TIEMPOS DE PROCESO DE LOS
TRABAJOS EN TODAS LAS MAQUINAS*/

  VECTOR_LONG TIEMPOS_TOTALES=DIM_VECTOR_LONG(n);
  int i=0;
  int j=0;
  for(i=0;i<(n);i++)
  {
    TIEMPOS_TOTALES[i]=0;
  }

  for(i=0;i<(n);i++)
  {
    for(j=0;j<(m);j++)
```

```
        {
           TIEMPOS_TOTALES[i]=TIEMPOS_TOTALES[i]+pt[i][j];
        }
    }

/*SE ORDENA SEGÚN LA REGLA LPT TODOS LOS TRABAJOS*/

    VECTOR_INT LPT=DIM_VECTOR_INT(n);
    VECTOR_LONG vector_copia=DIM_VECTOR_LONG(n);
    copy_vector(TIEMPOS_TOTALES,vector_copia,n);
    sort_vector(vector_copia,LPT,n,'D');
    printf("EL VECTOR ORDENADO ES");

/*CONSTRUCCIÓN DE LAS DOS PRIMERAS COMBINACIONES A EVALUAR*/
    int aux=2;
    VECTOR_INT COMIENZO=DIM_VECTOR_INT(aux);
    VECTOR_INT ALTERNATIVO=DIM_VECTOR_INT(aux);
    for(i=0;i<aux;i++)
       {
         COMIENZO[i]=LPT[i];
       }
    print_vector(COMIENZO,aux);
    ALTERNATIVO[0]=COMIENZO[1];
    ALTERNATIVO[1]=COMIENZO[0];

/*CONSTRUCCIÓN DEL MAKESPAN*/
    long int cm=0;
    long int cmx_1=0;
    long int cmx_11=0;
    cmx_1=pt[COMIENZO[0]][0];
    cmx_11=cmx_1+pt[COMIENZO[1]][0];
    for(j=1;j<m;j++){
          cmx_1=cmx_1+pt[COMIENZO[0]][j];
          if(cmx_1>=cmx_11)
             {
                cm=cmx_1+pt[COMIENZO[1]][j];
             }else
             {
```

```
                cm=cmx_11+pt[COMIENZO[1]][j];
            }
        cmx_11=cm;
    }

  cm=0;
  long int cmx_2=0;
  long int cmx_21=0;
  cmx_2=cmx_2+pt[ALTERNATIVO[0]][0];
  cmx_21=cmx_2+pt[ALTERNATIVO[1]][0];
  for(j=1;j<m;j++)
    {
      cmx_2=cmx_2+pt[ALTERNATIVO[0]][j];
       if(cmx_2>=cmx_21)
         {
           cm=cmx_2+pt[ALTERNATIVO[1]][j];
         }else
         {
           cm=cmx_21+pt[ALTERNATIVO[1]][j];
         }
      cmx_21=cm;
    }

/* SE EVALUA EL MAKESPAN PARA LAS DOS PRIMERAS SECUENCIAS PARCIALES POSIBLES*/

  VECTOR_INT secuencia_inicio=DIM_VECTOR_INT(n);
  if(cmx_11<cmx_21){
      copy_vector(COMIENZO,secuencia_inicio,aux);
  }else{
      copy_vector(ALTERNATIVO,secuencia_inicio,aux);
  }
 print_vector(secuencia_inicio,aux);

/*ITERACIONES DEL ALGORITMO NEH PARA TODO EL CONJUNTO DE TRABAJOS DE LOS DOS
GRUPOS*/

  int w=0;
  int p=0;
```

```
int r=0;
int c=0;
int d=0;
float alfa=0.4;/*A ELECCIÓN*/
long int makesp1=0;
long int makesp2A=500000000;
float epsilonTOTAL=0;
long int mksB=0;
long int mksA=0;
epsilonTOTAL=epsilon*(1-alfa);
int repeat=0;
for(c=3;c<=n;c++)/*AÑADIR UN NUEVO TRABAJO*/
{   int k=1;
    int fac=0;
    int bandera=0;
    int makesp2A=500000000;
    VECTOR_LONG TIEMPO=DIM_VECTOR_LONG(m);
    VECTOR_INT sec_d=DIM_VECTOR_INT(c);
    VECTOR_INT posicionesB_S=DIM_VECTOR_INT(c-1);
    VECTOR_INT valoresB_S=DIM_VECTOR_INT(c-1);
    VECTOR_INT repeticiones=DIM_VECTOR_INT(1);
    repeticiones[0]=0;
    int cont=0;
    for(r=0;r<c-1;r++)
        {
          if(secuencia_inicio[r]>(n/2)-1)
          {
             posicionesB_S[cont]=r;
             valoresB_S[cont]=secuencia_inicio[r];
             cont=cont+1;
          }
        }
    while(bandera==0)
    {
    for(p=0;p<c;p++)
    {/*INSERTAR EL NUEVO TRABAJO EN LAS DIFERENTES POSICIONES*/
       insert_vector(secuencia_inicio,c,LPT[c-1], p);
       long int auxiliarB=0;
```

```
long int auxiliarA=0;
long int makespanB=0;
long int makespanA=0;
for(r=0;r<c;r++)
   {
      if(secuencia_inicio[r]>(n/2)-1)
        {
          auxiliarB=r;
        }
   }
for(r=0;r<c;r++)
   {
      if(secuencia_inicio[r]<=(n/2)-1)
        {
          auxiliarA=r;
        }
   }
long int sum=0;
for(r=0;r<m;r++)
   {
   TIEMPO[r]=sum+pt[secuencia_inicio[0]][r];
   sum=TIEMPO[r];
   }
/*BUCLE PARA BUSCAR EL MENOR MAKESPAN*/
for(i=1;i<c;i++)
   {
      TIEMPO[0]=TIEMPO[0]+pt[secuencia_inicio[i]][0];

      for(w=1; w<m; w++)
      {   /*printf("m ");*/
         if (TIEMPO[w]>=TIEMPO[w-1])
         {
            TIEMPO[w]=TIEMPO[w]+pt[secuencia_inicio[i]][w];
         }
         if(TIEMPO[w]<TIEMPO[w-1])
         {
            TIEMPO[w]=TIEMPO[w-1]+pt[secuencia_inicio[i]][w];
         }
```

```
                /*print_vector(TIEMPO,m);*/
            }
            if(i==auxiliarB)
               {   /*GUARDAR EL VALOR DEL MAKESPAN DE B*/
                  makespanB=TIEMPO[w-1];
               }
            if(i==auxiliarA)
               {   /*GUARDAR EL VALOR DEL MAKESPAN DE A*/
                  makespanA=TIEMPO[w-1];
               }
         }
      makesp1=TIEMPO[m-1];
     /*SE CUMPLE LA RESTRICCIÓN DEL PROBLEMA MINIMIZANDO EL MAKESPAN DE A*/
      if(makespanA<=makesp2A && makespanB<=epsilonTOTAL)
      {
         copy_vector(secuencia_inicio,sec_d,c);
         makesp2A=makespanA;
         mksA=makespanA;
         mksB=makespanB;
         fac=1;
      }
      extract_vector(secuencia_inicio,c,p);
      /*SI NO EXISTE NINGUNA SECUENCIA PARCIAL FACTIBLE*/
      if(p==c-1 && fac==0)
      {   VECTOR_INT secuencia=DIM_VECTOR_INT(c);
         /*SE HACE USO DE LA FUNCIÓN REPARACIÓN*/
       secuencia=funcionreparacion(repeticiones, secuencia_inicio,valoresB_S, posicionesB_S, k, c, n, m,
pt, epsilonTOTAL, LPT);
         copy_vector(secuencia,secuencia_inicio,c);
         makesp2A=500000000;
         if(repeticiones[0]>=n){
               copy_vector(secuencia_inicio,sec_d,c);
               bandera=1;
               repeat=1;}

      }
      /*SALE DEL BUCLE CUANDO LA SECUENCIA PARCIAL ES FACTIBLE*/
      if(fac==1 && p==c-1)
```

```
            {
               bandera=1;
            }
         }
         }
          copy_vector(sec_d,secuencia_inicio,c);
         /* printf("el makespan de A: %ld\n",makesp2A);*/
      }
      if(repeat==0)
      {
      printf("el makespan final de A para esta secuencia es: %ld\n",mksA);
      printf("el makespan final de B para esta secuencia es: %ld\n",mksB);
      print_vector(secuencia_inicio,c-1);
      }
      if (repeat==1)
      {
         printf("no hay solucion factible para este problema\n");
      }
/*GUARDAR DATOS DE SALIDA*/
      FILE* fichero;
      fichero = fopen("salidaepsilonNEH.txt", "a");
      fprintf (fichero, "%ld\n", mksB);
      fprintf (fichero, "%ld\n", mksA, '\n');
      fprintf (fichero, "%d\n", repeat, '\n');
      fclose(fichero);
      printf("Proceso completado");

/*LIBERAR MEMORIA*/
free_mat_long(pt,n);
free(secuencia_inicio);
free(TIEMPOS_TOTALES);
free(LPT);
free(vector_copia);
free(COMIENZO);
free(ALTERNATIVO);
free(secuencia_inicio);
free(makesp2A);
free(mksA);
```

```
free(mksB);


    return 0;
}
```

Anexo D

Código algoritmo NEH adaptado a dos grupos de trabajos con la regla de desempate propuesta, $NEH_{PROPUESTA}$

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
VECTOR_INT funcionreparacion(VECTOR_INT repeticiones, VECTOR_INT secuencia_inicio, VECTOR_INT valoresB_S, VECTOR_INT posicionesB_S, int k, int c, int n, int m, MAT_LONG pt, long int epsilonTOTAL, VECTOR_INT LPT);
int main(int argc, char*argv[])
{
/*ESTIMACIÓN VALOR INICIAL DE EPSILON*/

  printf("VAMOS A USAR EL METODO NEH PARA MODELAR LOS DOS GRUPOS DE TRABAJOS\n");

/*FUNCIÓN LECTURA DE FICHEROS*/

FILE *archivo;
int temporal;
MAT_LONG pt;
int n=0;
int m=0;
int pi=0;
int pj=0;
int epsilon=0;

archivo = fopen(argv[1],"rt");
if (archivo == NULL)
    {
      printf("\n Error de apertura del archivo \n\n");
    }
    else
    {
      printf("\n El contenido del archivo de prueba es \n\n");
      fscanf(archivo,"%d\n", &temporal);
```

```
        m=temporal;
        printf("el n de maquinas es %d\n",m);

        fscanf(archivo,"%d\n", &temporal);
        n=temporal;
        printf("el n de trabajos es %d\n",n);

        fscanf(archivo,"%d\n", &temporal);
        epsilon=temporal;
        printf("el epsilon es %d\n",epsilon);
        pt=DIM_MAT_LONG(n,m);

        for(pi=0;pi<n;pi++)
        {
        for(pj=0;pj<m;pj++)
            {
            fscanf(archivo,"%10d", &temporal);
            pt[pi][pj]=temporal;
            }
        }

    }

    fclose(archivo);

/*PARA ESTE PROBLEMA SE TOMARÁ EL TOTAL DE LOS TIEMPOS DE PROCESO DE LOS
TRABAJOS EN TODAS LAS MAQUINAS*/

  VECTOR_LONG TIEMPOS_TOTALES=DIM_VECTOR_LONG(n);
  int i=0;
  int j=0;
  for(i=0;i<(n);i++)
  {
    TIEMPOS_TOTALES[i]=0;
  }

  for(i=0;i<(n);i++)
  {
```

```
        for(j=0;j<(m);j++)
            {
                TIEMPOS_TOTALES[i]=TIEMPOS_TOTALES[i]+pt[i][j];
            }
    }

/*SE ORDENA SEGÚN LA REGLA LPT TODOS LOS TRABAJOS*/

    VECTOR_INT LPT=DIM_VECTOR_INT(n);
    VECTOR_LONG vector_copia=DIM_VECTOR_LONG(n);
    copy_vector(TIEMPOS_TOTALES,vector_copia,n);
    sort_vector(vector_copia,LPT,n,'D');
    printf("EL VECTOR ORDENADO ES");

/*CONSTRUCCIÓN DE LAS DOS PRIMERAS COMBINACIONES A EVALUAR*/
    int aux=2;
    VECTOR_INT COMIENZO=DIM_VECTOR_INT(aux);
    VECTOR_INT ALTERNATIVO=DIM_VECTOR_INT(aux);
    for(i=0;i<aux;i++)
        {
          COMIENZO[i]=LPT[i];
        }
    print_vector(COMIENZO,aux);
    ALTERNATIVO[0]=COMIENZO[1];
    ALTERNATIVO[1]=COMIENZO[0];

/*CONSTRUCCIÓN DEL MAKESPAN*/
    long int cm=0;
    long int cmx_1=0;
    long int cmx_11=0;
    cmx_1=pt[COMIENZO[0]][0];
    cmx_11=cmx_1+pt[COMIENZO[1]][0];
    for(j=1;j<m;j++){
          cmx_1=cmx_1+pt[COMIENZO[0]][j];
          if(cmx_1>=cmx_11)
              {
                  cm=cmx_1+pt[COMIENZO[1]][j];
              }else
```

```
            {
               cm=cmx_11+pt[COMIENZO[1]][j];
            }
         cmx_11=cm;
      }

   cm=0;
   long int cmx_2=0;
   long int cmx_21=0;
   cmx_2=cmx_2+pt[ALTERNATIVO[0]][0];
   cmx_21=cmx_2+pt[ALTERNATIVO[1]][0];
   for(j=1;j<m;j++)
      {
         cmx_2=cmx_2+pt[ALTERNATIVO[0]][j];
          if(cmx_2>=cmx_21)
            {
               cm=cmx_2+pt[ALTERNATIVO[1]][j];
            }else
            {
               cm=cmx_21+pt[ALTERNATIVO[1]][j];
            }
         cmx_21=cm;
      }

/* SE EVALUA EL MAKESPAN PARA LAS DOS PRIMERAS SECUENCIAS PARCIALES POSIBLES*/

   VECTOR_INT secuencia_inicio=DIM_VECTOR_INT(n);
   if(cmx_11<cmx_21){
         copy_vector(COMIENZO,secuencia_inicio,aux);
   }else{
         copy_vector(ALTERNATIVO,secuencia_inicio,aux);
   }
  /*print_vector(secuencia_inicio,aux);*/

/*ITERACIONES DEL ALGORITMO NEH PARA TODO EL CONJUNTO DE TRABAJOS DE LOS DOS
GRUPOS*/

   int w=0;
```

```
int p=0;
int r=0;
int c=0;
int d=0;
float alfa=0;/*A ELECCIÓN*/
long int makesp1=0;
long int makesp2A=500000000;
float epsilonTOTAL=0;
long int mksB=0;
long int mksA=0;
epsilonTOTAL=epsilon*(1-alfa);
int repeat=0;
for(c=3;c<=n;c++)/*AÑADIR UN NUEVO TRABAJO*/
{   int k=1;
    int fac=0;
    int bandera=0;
    int makesp2A=500000000;
    VECTOR_LONG TIEMPO=DIM_VECTOR_LONG(m);
    VECTOR_INT sec_d=DIM_VECTOR_INT(c);
    VECTOR_INT posicionesB_S=DIM_VECTOR_INT(c-1);
    VECTOR_INT valoresB_S=DIM_VECTOR_INT(c-1);
    VECTOR_INT repeticiones=DIM_VECTOR_INT(1);
    repeticiones[0]=0;
    int cont=0;
    for(r=0;r<c-1;r++)
        {
          if(secuencia_inicio[r]>(n/2)-1)
          {
             posicionesB_S[cont]=r;
             valoresB_S[cont]=secuencia_inicio[r];
             cont=cont+1;
          }
        }
    while(bandera==0)
    {
    for(p=0;p<c;p++)
    {/*INSERTAR EL NUEVO TRABAJO EN LAS DIFERENTES POSICIONES*/
       insert_vector(secuencia_inicio,c,LPT[c-1], p);
```

```
long int auxiliarB=0;
long int auxiliarA=0;
long int makespanB=0;
long int makespanA=0;
for(r=0;r<c;r++)
   {
      if(secuencia_inicio[r]>(n/2)-1)
        {
          auxiliarB=r;
        }
   }
for(r=0;r<c;r++)
   {
      if(secuencia_inicio[r]<=(n/2)-1)
        {
          auxiliarA=r;
        }
   }
long int sum=0;
for(r=0;r<m;r++)
   {
   TIEMPO[r]=sum+pt[secuencia_inicio[0]][r];
   sum=TIEMPO[r];
   }
/*BUCLE PARA BUSCAR EL MENOR MAKESPAN*/
for(i=1;i<c;i++)
   {
      TIEMPO[0]=TIEMPO[0]+pt[secuencia_inicio[i]][0];

      for(w=1; w<m; w++)
      {  /*printf("m ");*/
         if (TIEMPO[w]>=TIEMPO[w-1])
         {
            TIEMPO[w]=TIEMPO[w]+pt[secuencia_inicio[i]][w];
         }
         if(TIEMPO[w]<TIEMPO[w-1])
         {
            TIEMPO[w]=TIEMPO[w-1]+pt[secuencia_inicio[i]][w];
```

```
            }
            /*print_vector(TIEMPO,m);*/
        }
        if(i==auxiliarB)
            {   /*GUARDAR EL VALOR DEL MAKESPAN DE B*/
                makespanB=TIEMPO[w-1];
            }
        if(i==auxiliarA)
            {   /*GUARDAR EL VALOR DEL MAKESPAN DE A*/
                makespanA=TIEMPO[w-1];
            }
    }
 makesp1=TIEMPO[m-1];
/*SE CUMPLE LA RESTRICCIÓN DEL PROBLEMA MINIMIZANDO EL MAKESPAN DE A*/
 if(makespanA<=makesp2A && makespanB<=epsilonTOTAL)
 {   if(makespanA==makesp2A)
    {   if(makespanB<=mksB)
        {
            copy_vector(secuencia_inicio,sec_d,c);
            makesp2A=makespanA;
            mksA=makespanA;
            mksB=makespanB;
            fac=1;
        }
    }else{
        copy_vector(secuencia_inicio,sec_d,c);
        makesp2A=makespanA;
        mksA=makespanA;
        mksB=makespanB;
        fac=1;
    }

 }
 extract_vector(secuencia_inicio,c,p);
 /*SI NO EXISTE NINGUNA SECUENCIA PARCIAL FACTIBLE*/
 if(p==c-1 && fac==0)
 {   VECTOR_INT secuencia=DIM_VECTOR_INT(c);
    /*SE HACE USO DE LA FUNCIÓN REPARACIÓN*/
```

```
            secuencia=funcionreparacion(repeticiones, secuencia_inicio,valoresB_S, posicionesB_S, k, c, n, m,
pt, epsilonTOTAL, LPT);
                copy_vector(secuencia,secuencia_inicio,c);
                makesp2A=500000000;
                if(repeticiones[0]>=n){
                        copy_vector(secuencia_inicio,sec_d,c);
                        bandera=1;
                        repeat=1;}


            }
            /*SALE DEL BUCLE CUANDO LA SECUENCIA PARCIAL ES FACTIBLE*/
            if(fac==1 && p==c-1)
            {
               bandera=1;
            }
        }
        }
         copy_vector(sec_d,secuencia_inicio,c);
        /* printf("el makespan de A: %ld\n",makesp2A);*/
    }
    if(repeat==0)
    {
    printf("el makespan final de A para esta secuencia es: %ld\n",mksA);
    printf("el makespan final de B para esta secuencia es: %ld\n",mksB);
    print_vector(secuencia_inicio,c-1);
    }
    if (repeat==1)
    {
       printf("no hay solucion factible para este problema\n");
    }
/*GUARDAR DATOS DE SALIDA*/
    FILE* fichero;
    fichero = fopen("sol_desempate_00.txt", "a");
    fprintf (fichero, "%ld\n", mksB);
    fprintf (fichero, "%ld\n", mksA, '\n');
    fprintf (fichero, "%d\n", repeat, '\n');
    fclose(fichero);
    printf("Proceso completado");
```

```
/*LIBERAR MEMORIA*/
free_mat_long(pt,n);
free(secuencia_inicio);
free(TIEMPOS_TOTALES);
free(LPT);
free(vector_copia);
free(COMIENZO);
free(ALTERNATIVO);
free(secuencia_inicio);
free(makesp2A);
free(mksA);
free(mksB);


   return 0;
}
```

Anexo E

Código algoritmo NEH adaptado a dos grupos de trabajos con la regla de desempate FF, NEH_{FF}

```
#include <schedule.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
VECTOR_INT funcionreparacion(VECTOR_INT repeticiones, VECTOR_INT secuencia_inicio, VECTOR_INT valoresB_S, VECTOR_INT posicionesB_S, int k, int c, int n, int m, MAT_LONG pt, long int epsilonTOTAL, VECTOR_INT LPT);
int main(int argc, char*argv[])
{
/*ESTIMACIÓN VALOR INICIAL DE EPSILON*/

  printf("VAMOS A USAR EL METODO NEH PARA MODELAR LOS DOS GRUPOS DE TRABAJOS\n");

/*FUNCIÓN LECTURA DE FICHEROS*/

FILE *archivo;
int temporal;
MAT_LONG pt;
int n=0;
int m=0;
int pi=0;
int pj=0;
int epsilon=0;

archivo = fopen(argv[1],"rt");
if (archivo == NULL)
      {
         printf("\n Error de apertura del archivo \n\n");
      }
      else
      {
         printf("\n El contenido del archivo de prueba es \n\n");
         fscanf(archivo,"%d\n", &temporal);
         m=temporal;
```

```
        printf("el n de maquinas es %d\n",m);

        fscanf(archivo,"%d\n", &temporal);
        n=temporal;
        printf("el n de trabajos es %d\n",n);

        fscanf(archivo,"%d\n", &temporal);
        epsilon=temporal;
        printf("el epsilon es %d\n",epsilon);
        pt=DIM_MAT_LONG(n,m);

        for(pi=0;pi<n;pi++)
        {
        for(pj=0;pj<m;pj++)
            {
            fscanf(archivo,"%10d", &temporal);
            pt[pi][pj]=temporal;
            }
        }

    }

    fclose(archivo);

/*PARA ESTE PROBLEMA SE TOMARÁ EL TOTAL DE LOS TIEMPOS DE PROCESO DE LOS
TRABAJOS EN TODAS LAS MAQUINAS*/

  VECTOR_LONG TIEMPOS_TOTALES=DIM_VECTOR_LONG(n);
  int i=0;
  int j=0;
  for(i=0;i<(n);i++)
  {
    TIEMPOS_TOTALES[i]=0;
  }

  for(i=0;i<(n);i++)
  {
    for(j=0;j<(m);j++)
```

```
        {
           TIEMPOS_TOTALES[i]=TIEMPOS_TOTALES[i]+pt[i][j];
        }
    }

/*SE ORDENA SEGÚN LA REGLA LPT TODOS LOS TRABAJOS*/

    VECTOR_INT LPT=DIM_VECTOR_INT(n);
    VECTOR_LONG vector_copia=DIM_VECTOR_LONG(n);
    copy_vector(TIEMPOS_TOTALES,vector_copia,n);
    sort_vector(vector_copia,LPT,n,'D');
    printf("EL VECTOR ORDENADO ES");

/*CONSTRUCCIÓN DE LAS DOS PRIMERAS COMBINACIONES A EVALUAR*/
    int aux=2;
    VECTOR_INT COMIENZO=DIM_VECTOR_INT(aux);
    VECTOR_INT ALTERNATIVO=DIM_VECTOR_INT(aux);
    for(i=0;i<aux;i++)
       {
         COMIENZO[i]=LPT[i];
       }
    print_vector(COMIENZO,aux);
    ALTERNATIVO[0]=COMIENZO[1];
    ALTERNATIVO[1]=COMIENZO[0];

/*CONSTRUCCIÓN DEL MAKESPAN*/
    long int cm=0;
    long int cmx_1=0;
    long int cmx_11=0;
    cmx_1=pt[COMIENZO[0]][0];
    cmx_11=cmx_1+pt[COMIENZO[1]][0];
    for(j=1;j<m;j++){
          cmx_1=cmx_1+pt[COMIENZO[0]][j];
          if(cmx_1>=cmx_11)
             {
                cm=cmx_1+pt[COMIENZO[1]][j];
             }else
             {
```

```
                cm=cmx_11+pt[COMIENZO[1]][j];
            }
        cmx_11=cm;
    }

cm=0;
long int cmx_2=0;
long int cmx_21=0;
cmx_2=cmx_2+pt[ALTERNATIVO[0]][0];
cmx_21=cmx_2+pt[ALTERNATIVO[1]][0];
for(j=1;j<m;j++)
    {
        cmx_2=cmx_2+pt[ALTERNATIVO[0]][j];
         if(cmx_2>=cmx_21)
            {
                cm=cmx_2+pt[ALTERNATIVO[1]][j];
            }else
            {
                cm=cmx_21+pt[ALTERNATIVO[1]][j];
            }
        cmx_21=cm;
    }

/* SE EVALUA EL MAKESPAN PARA LAS DOS PRIMERAS SECUENCIAS PARCIALES POSIBLES*/

VECTOR_INT secuencia_inicio=DIM_VECTOR_INT(n);
if(cmx_11<cmx_21){
        copy_vector(COMIENZO,secuencia_inicio,aux);
}else{
        copy_vector(ALTERNATIVO,secuencia_inicio,aux);
}
 /*print_vector(secuencia_inicio,aux);*/

/*ITERACIONES DEL ALGORITMO NEH PARA TODO EL CONJUNTO DE TRABAJOS DE LOS DOS
GRUPOS*/

int w=0;
int p=0;
```

```
int r=0;
int c=0;
int d=0;
long int itA=0;
long int itAt=0;
long int it=0;
float alfa=0;/*A ELECCIÓN*/
long int makesp1=0;
long int makesp2A=500000000;
float epsilonTOTAL=0;
long int mksB=0;
long int mksA=0;
epsilonTOTAL=epsilon*(1-alfa);
int repeat=0;
for(c=3;c<=n;c++)/*AÑADIR UN NUEVO TRABAJO*/
{   int k=1;
    int fac=0;
    int bandera=0;
    int makesp2A=500000000;
    VECTOR_LONG TIEMPO=DIM_VECTOR_LONG(m);
    VECTOR_INT sec_d=DIM_VECTOR_INT(c);
    VECTOR_INT posicionesB_S=DIM_VECTOR_INT(c-1);
    VECTOR_INT valoresB_S=DIM_VECTOR_INT(c-1);
    VECTOR_INT repeticiones=DIM_VECTOR_INT(1);
    repeticiones[0]=0;
    int cont=0;
    for(r=0;r<c-1;r++)
        {
            if(secuencia_inicio[r]>(n/2)-1)
            {
                posicionesB_S[cont]=r;
                valoresB_S[cont]=secuencia_inicio[r];
                cont=cont+1;
            }
        }
    while(bandera==0)
    {
    for(p=0;p<c;p++)
```

```
{/*INSERTAR EL NUEVO TRABAJO EN LAS DIFERENTES POSICIONES*/
   insert_vector(secuencia_inicio,c,LPT[c-1], p);
   long int auxiliarB=0;
   long int auxiliarA=0;
   long int makespanB=0;
   long int makespanA=0;
   for(r=0;r<c;r++)
      {
         if(secuencia_inicio[r]>(n/2)-1)
            {
               auxiliarB=r;
            }
      }
   for(r=0;r<c;r++)
      {
         if(secuencia_inicio[r]<=(n/2)-1)
            {
               auxiliarA=r;
            }
      }
   long int sum=0;
   for(r=0;r<m;r++)
      {
      TIEMPO[r]=sum+pt[secuencia_inicio[0]][r];
      sum=TIEMPO[r];
      }
   /*BUCLE PARA BUSCAR EL MENOR MAKESPAN*/
   for(i=1;i<c;i++)
      {
         TIEMPO[0]=TIEMPO[0]+pt[secuencia_inicio[i]][0];

         for(w=1; w<m; w++)
         {   /*printf("m ");*/
            if (TIEMPO[w]>=TIEMPO[w-1])
            {
               TIEMPO[w]=TIEMPO[w]+pt[secuencia_inicio[i]][w];
            }
            if(TIEMPO[w]<TIEMPO[w-1])
```

```
            {
               TIEMPO[w]=TIEMPO[w-1]+pt[secuencia_inicio[i]][w];
            }
            /*print_vector(TIEMPO,m);*/
         }
         if(i==auxiliarB)
            {   /*GUARDAR EL VALOR DEL MAKESPAN DE B*/
               makespanB=TIEMPO[w-1];
            }
         if(i==auxiliarA)
            {   /*GUARDAR EL VALOR DEL MAKESPAN DE A*/
               makespanA=TIEMPO[w-1];
            }
      }
   makesp1=TIEMPO[m-1];
  /*SE CUMPLE LA RESTRICCIÓN DEL PROBLEMA MINIMIZANDO EL MAKESPAN DE A*/
   if(makespanA<makesp2A && makespanB<=epsilonTOTAL)
   {
      copy_vector(secuencia_inicio,sec_d,c);
      makesp2A=makespanA;
      mksA=makespanA;
      mksB=makespanB;
      it=itAt;
      fac=1;
   }
   for(w=1; w<m; w++)
   {  for (r=0;r<c;r++)
      {
         itA=itA+pt[r][w];
      }
     itAt=itAt+(TIEMPO[w]-itA);
     itA=0;
   }
   if(makespanA==makesp2A && makespanB<=epsilonTOTAL)
   {if(itAt<=it){
      copy_vector(secuencia_inicio,sec_d,c);
      makesp2A=makespanA;
      mksA=makespanA;
```

```
            mksB=makespanB;
            it=itAt;
            fac=1;}
        }
        extract_vector(secuencia_inicio,c,p);
        itA=0;
        itAt=0;
        /*SI NO EXISTE NINGUNA SECUENCIA PARCIAL FACTIBLE*/
        if(p==c-1 && fac==0)
        {   VECTOR_INT secuencia=DIM_VECTOR_INT(c);
            /*SE HACE USO DE LA FUNCIÓN REPARACIÓN*/
          secuencia=funcionreparacion(repeticiones, secuencia_inicio,valoresB_S, posicionesB_S, k, c, n, m,
pt, epsilonTOTAL, LPT);
            copy_vector(secuencia,secuencia_inicio,c);
            makesp2A=500000000;
            if(repeticiones[0]>=n){
                copy_vector(secuencia_inicio,sec_d,c);
                bandera=1;
                repeat=1;}

        }
        /*SALE DEL BUCLE CUANDO LA SECUENCIA PARCIAL ES FACTIBLE*/
        if(fac==1 && p==c-1)
        {
            bandera=1;
        }
      }
      }
       copy_vector(sec_d,secuencia_inicio,c);
      /* printf("el makespan de A: %ld\n",makesp2A);*/
   }
   if(repeat==0)
   {
   printf("el makespan final de A para esta secuencia es: %ld\n",mksA);
   printf("el makespan final de B para esta secuencia es: %ld\n",mksB);
   print_vector(secuencia_inicio,c-1);
   }
   if (repeat==1)
```

```
    {
        printf("no hay solucion factible para este problema\n");
    }
/*GUARDAR DATOS DE SALIDA*/
    FILE* fichero;
    fichero = fopen("sol_nehff_00.txt", "a");
    fprintf (fichero, "%ld\n", mksB);
    fprintf (fichero, "%ld\n", mksA, '\n');
    fprintf (fichero, "%d\n", repeat, '\n');
    fclose(fichero);
    printf("Proceso completado");

/*LIBERAR MEMORIA*/
free_mat_long(pt,n);
free(secuencia_inicio);
free(TIEMPOS_TOTALES);
free(LPT);
free(vector_copia);
free(COMIENZO);
free(ALTERNATIVO);
free(secuencia_inicio);
free(makesp2A);
free(mksA);
free(mksB);


    return 0;
}
```

Anexo F

Código Función Reparación

Declaración de la función:

VECTOR_INT **funcionreparacion**(VECTOR_INT repeticiones, VECTOR_INT secuencia_inicio, VECTOR_INT valoresB_S, VECTOR_INT posicionesB_S, int k, int c, int n, int m, MAT_LONG pt, long int epsilonTOTAL, VECTOR_INT LPT);

Desarrollo de la función:

```
VECTOR_INT funcionreparacion(VECTOR_INT repeticiones, VECTOR_INT secuencia_inicio, VECTOR_INT valoresB_S, VECTOR_INT posicionesB_S, int k, int c, int n, int m, MAT_LONG pt, long int epsilonTOTAL, VECTOR_INT LPT)
{
int posicionauxiliar=0;
int valorauxiliar=0;
int r=0;
int i=0;
int a=0;
int w=0;
int d=c-1;
int aux=0;
/*VECTORES AUXILIARES NECESARIOS PARA EL ALGORITMO*/
   VECTOR_INT posmaximoB=DIM_VECTOR_INT(d);
   VECTOR_INT posicionesB=DIM_VECTOR_INT(d);
   VECTOR_INT posicionesB_n=DIM_VECTOR_INT(d);
   VECTOR_INT valoresB_n=DIM_VECTOR_INT(d);
/*SE GUARDA LA POSICIÓN Y EL VALOR DEL ÚLTIMO TRABAJO DE B */
   int cont=0;
   for(r=0;r<d;r++)
      {
         if(secuencia_inicio[r]>(n/2)-1)
         {
            posicionesB[cont]=r;
            cont=cont+1;
         }
      }
   posicionauxiliar=posicionesB[cont-1];
   valorauxiliar=secuencia_inicio[posicionauxiliar];
   extract_vector(secuencia_inicio,d,posicionauxiliar);
   aux=posicionesB_S[search_vector(valoresB_S,cont,valorauxiliar)];
```

```
/*COMENZAMOS EL ALGORITMO NEH PARA DESPLAZAR HACIA LA DERECHA EL TRABAJO DE B*/
   int fac=0;
   int p=0;
   long int makesp2A=500000000;
   VECTOR_LONG TIEMPO=DIM_VECTOR_LONG(m);
   VECTOR_INT sec_d=DIM_VECTOR_INT(d);
   /*SE COMPROBARÁN TODAS LAS POSICIONES A LA IZQUIERDA DE LA ACTUAL*/
   for(p=0;p<aux;p++)
        {/*INSERTAR EL SIGUIENTE TRABAJO EN LAS DIFERENTES POSICIONES*/
          insert_vector(secuencia_inicio,d,valorauxiliar, p);
          long int auxiliarB=0;
          long int auxiliarA=0;
          long int makespanB=0;
          long int makespanA=0;
          for(r=0;r<d;r++)
             {
                if(secuencia_inicio[r]>(n/2)-1)
                   {
                      auxiliarB=r;
                   }
             }
          for(r=0;r<d;r++)
             {
                if(secuencia_inicio[r]<=(n/2)-1)
                   {
                      auxiliarA=r;
                   }
             }
          long int sum=0;
          for(r=0;r<m;r++)
             {
             TIEMPO[r]=sum+pt[secuencia_inicio[0]][r];
             sum=TIEMPO[r];
             }
          /*BUCLE PARA BUSCAR EL MENOR MAKESPAN*/
          for(i=1;i<d;i++)
```

```
        {
            TIEMPO[0]=TIEMPO[0]+pt[secuencia_inicio[i]][0];

            for(w=1; w<m; w++)
            {  if (TIEMPO[w]>=TIEMPO[w-1])
                {
                    TIEMPO[w]=TIEMPO[w]+pt[secuencia_inicio[i]][w];
                }
                if(TIEMPO[w]<TIEMPO[w-1])
                {
                    TIEMPO[w]=TIEMPO[w-1]+pt[secuencia_inicio[i]][w];
                }
            }
            if(i==auxiliarB)
                {   /*GUARDAR EL VALOR DEL MAKESPAN DE B*/
                    makespanB=TIEMPO[w-1];
                }
            if(i==auxiliarA)
                {   /*GUARDAR EL VALOR DEL MAKESPAN DE A*/
                    makespanA=TIEMPO[w-1];
                }
        }

    long int  makesp1=TIEMPO[m-1];
    if(makespanA<=makesp2A && makespanB<=epsilonTOTAL)
    {
        copy_vector(secuencia_inicio,sec_d,d);
        makesp2A=makespanA;
        fac=1;
    }
    extract_vector(secuencia_inicio,d,p);
}
 if(p==(aux) && aux==0 && fac==0)
    {   repeticiones[0]=repeticiones[0]+1;
        insert_vector(secuencia_inicio,d,valorauxiliar, aux);
        copy_vector(secuencia_inicio,sec_d,d);
        print_vector(secuencia_inicio,d);
    }
```

```
        if(p==(aux) && fac==0 && aux!=0)
            {
                insert_vector(secuencia_inicio,d,valorauxiliar, aux-1);
               /* print_vector(secuencia_inicio,d);*/
                copy_vector(secuencia_inicio,sec_d,d);
            }

        copy_vector(sec_d,secuencia_inicio,d);

        /*SE GUARDAN LAS POSICIONES ACTUALES DE LOS VALORES DE B*/
        int contn=0;
        for(r=0;r<d;r++)
        {
            if(secuencia_inicio[r]>(n/2)-1)
            {
                posicionesB_n[contn]=r;
                valoresB_n[contn]=secuencia_inicio[r];
                contn=contn+1;
            }
        }
       /*SE GUARDA LA MENOR POSICIÓN PARA ESTE TRABAJO ENTRE LA SECUENCIA
HALLADA Y LA SECUENCIA PARCIAL ORIGINAL*/

if(posicionesB_S[search_vector(valoresB_S,cont,valorauxiliar)]>=posicionesB_n[search_vector(valor
esB_n,cont,valorauxiliar)])
            {

posicionesB_S[search_vector(valoresB_S,cont,valorauxiliar)]=posicionesB_n[search_vector(valoresB
_n,cont,valorauxiliar)];
            }


    /*GUARDAMOS LA SECUENCIA REPARADA*/
    VECTOR_INT sec_reparada=DIM_VECTOR_INT(c);
    copy_vector(secuencia_inicio,sec_reparada,c);
    return sec_reparada;
}
```